MASTER EN SCIENCES INFORMATIQUES

Dialogue de commande H-M finalisé
représentation temporelle des connaissances et planification dans l'interface

Vandenhende, Frédéric

Award date:
1993

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix
Institut d'Informatique
Rue Grandgagnage, 21b
B-5000 NAMUR

Dialogue de commande H-M finalisé : représentation temporelle des connaissances et planification dans l'interface.

Frédéric Vandenhende

Promoteur : Jacques Berleur, s.j.

Mémoire présenté en vue
de l'obtention du grade de
Licencié et Maître en Informatique

Année académique 1992 - 1993

Merci ...

Merci d'abord à J.M. Pierrel, L. Romary et B. Gaiffe qui m'ont gentiment accueilli dans l'équipe dialogue du CRIN. Grâce à leurs conseils, ils ont rendu mon stage le plus agréable possible et m'ont fait découvrir un thème de recherche passionnant. Ce fût une joie de travailler avec vous.

Merci ensuite au Père Berleur, qui a su trouver le temps de m'écouter et me guider malgré la lourde charge de sa fonction.

Merci enfin à Pascal et à Frédéric qui m'ont permis de passer cinq mois en France que je ne suis pas prêt d'oublier.

Abstract

One of the main problems in the computer analysis of natural language is understanding utterances beyond a surface level. At a first level, a natural language understanding system which aims to command an application, can answer simple and trivial requests : it simply links action of the utterance to action of the application.

In order to extend the number, complexity and domain of possible commands it can process, we must add a more extended representation of the reality (knowledge of the application, knowledge of the user, ...) to this interface.

Thanks to this extended knowledge, the system extracts the intentions contained in the user's utterances (speech acts) and, by plan inferencing, finds and executes a sequence of task actions, or re-conducts the dialogue.

We present an implementation of these concepts in the Multiworks project using a Temporal Model of informations, les Zones Temporelles.

Résumé

Un des problèmes principaux de la compréhension du Langage Naturel par la machine est de traiter les énoncés au delà du niveau de surface. La plupart des systèmes de compréhension du Langage Naturel dont le but est de commander une application, se contentent, jusqu'à présent, de relier l'action comprise dans l'énoncé (le prédicat) avec une action de la tâche; ce qui limite fortement le nombre et la complexité des requêtes permises et rend le tout fort statique.

Afin de dépasser ce point de vue, il est indispensable d'augmenter les connaissances sur la réalité dont dispose l'interface (modèle de la tâche, modèle de l'utilisateur, représentation des intentions, ...).

Grâce à ces connaissances, le système est en mesure de dégager les intentions de l'utilisateur contenues dans ses énoncés et trouver une séquence d'actions qui y répond planification) ou relancer le dialogue.

Nous présentons une implémentation de ces concepts pour le projet Multiworks, autour de la représentation temporelle des connaissances, les Zones Temporelles.

Table des matières

Introduction

Le Centre de Recherche en Informatique de Nancy (CRIN), par l'ensemble des recherches menées en son sein et tout particulièrement par l'équipe Dialogue, a acquis une solide expérience dans la reconnaissance de la parole par la machine et la mise au point de systèmes de **dialogue homme-machine** en **Langage Naturel**.

Différents projets se sont succédé au cours des dernières années : MYRTILLE I et II, DIAL, ... [Pierrel 87], [Haton 91]. Ils ont, en tentant de résoudre le problème de la **communication** orale homme-machine finalisée en Langage Naturel (LN), dégagé leurs limites et mis à jour d'autres questions fondamentales sur un sujet en plein développement.

Différents axes de recherches nous sont apparus au cours de nos travaux.

Plusieurs axes de recherches

Ces différents axes ont été abordés simultanément ou en parallèle tant les matières qu'ils traitent sont indissociables et complémentaires. Introduisons les, cependant, un par un pour mieux cerner leurs apports respectifs.

Recherches sur le LN

Les différents travaux sur le Langage Naturel entrepris au CRIN ou parus dans les revues spécialisée, permettent de cerner les différents problèmes généraux que pose la communication homme-machine en "langage naturel". Nous les présenterons au Chapitre 1 et montrerons pourquoi on doit restreindre les ambitions de développer des systèmes de reconnaissance universelle du Langage Naturel et se contenter de systèmes locaux en Langage Naturel finalisé.

Linguistique et, plus précisément, philosophie du langage

La communication homme-machine en LN ne fonctionne que si la machine le reconnaît et le comprend.

La machine, pour décoder le signal de langage, passe par plusieurs niveaux de représentation. Le signal est tour à tour perçu comme une onde vocale, un treillis de phonèmes, un ensemble de mots, une construction grammaticale, ..., pour aboutir à une représentation du contenu informationnel de l'énoncé.

Le but des recherches sur le Langage Naturel est de définir des modèles associés à ces niveaux et de permettre la transition depuis le niveau du signal jusqu'au niveau supérieur [Mousel 90], [Deville 89].

Le contenu informationnel est obtenu en replaçant la sémantique du signal dans le contexte de la conversation (ce que l'on nomme la pragmatique [Levinson 83]).

On peut considérer que la machine a trouvé le sens d'un énoncé en situation dès qu'elle est arrivée à relier le signal avec les concepts de la réalité.

Les philosophes du langage (Austin, Searle, ...) ont élaboré une théorie selon laquelle tout énoncé du discours contenait un certain nombre d'actes ayant un impact direct sur la réalité (actes de langage). Nous nous intéresserons tout particulièrement à cette théorie (Chapitre 2) et verrons comment on peut la formaliser pour l'intégrer à la machine et quels sont les causes et conséquences de l'exécution de ces actes pour celle-ci.

Nous travaillerons donc tout particulièrement au niveau pragmatique; notre souci sera de spécifier cette charnière entre énoncé et réalité. Nous supposerons, pour ce faire, qu'il existe déjà des solutions satisfaisantes qui assurent le passage du signal vers les niveaux de représentation inférieurs.

Sciences cognitives

Le langage naturel est le moyen de communication de l'homme par excellence. Si on veut que la machine le comprenne, on est obligé de s'intéresser à la façon dont l'homme l'utilise et d'aborder le champ de recherches que l'on appelle aujourd'hui les "sciences cognitives".

Quels sont les connaissances impliquées dans la reconnaissance et la génération du signal et plus précisément, de quelles connaissances doit-on disposer pour faire le lien entre énoncés et réalité ([Mignot 92]) ?

En posant la question, on donne déjà des éléments de réponse. Pour faire le lien entre énoncés et réalité, il faut bien évidemment des connaissances sur les énoncés et sur la réalité.

- Connaissances sur les énoncés : pour la machine, ces connaissances proviennent du traitement successif du signal dans les différents niveaux (Cfr. supra et Chapitre 2).

- Connaissances sur la réalité : de quelle réalité parle-t-on ? Y a-t-il une corrélation entre réalité et langage ? Si oui, quelles sont les restrictions obligatoires que l'on doit porter à la réalité pour rendre opérationnel le système de reconnaissance du LN par la machine ? Le chapitre 1 amorcera des réponses à ces questions.

Lorsqu'on dispose de ces deux types de connaissances, il faut encore assurer le passage entre eux. L'utilisation d'une modélisation compatible passe par l'unification des représentations des énoncés et de la réalité.

Cette question a été brillamment soulevée par Laurent Romary : "*les informations manipulées dans un dialogue homme-machine ne doivent pas être disséminées en autant de modules indépendants* (comme c'était le cas avec les anciens systèmes) *mais intégrées pour faciliter les échanges dans une architecture cohérente tant au niveau de la langue qu'au niveau des concepts manipulés...*" ([Romary 89]). Sa thèse de doctorat et les développements du modèle présenté, le modèle des **Zones Temporelles**, ont contribué à apporter des solutions originales à cette problématique.

Ayant eu la chance de travailler avec L. Romary, c'est tout naturellement que nous avons utilisé les Zones Temporelles pour représenter les connaissances. Ce faisant, nous avons contribué à l'extension du modèle. Le lecteur trouvera aux chapitre 4 et 5 une présentation de cette modélisation.

Interface Homme-Machine

Les recherches sur le dialogue homme-machine en Langage Naturel font aussi appel évidemment à la conception de l'interface homme-machine. Cette discipline informatique, relativement récente, part de la constatation suivante : pour développer des applications informatiques portables et conviviales, il est indispensable, qu'à l'intérieur de la machine, on sépare les aspects "communication" des autres aspects techniques.

La machine est alors décomposée en deux : une entité qui réalise des traitements (SGBD, compilateur, outils de simulation, ...) et une qui s'occupe de faire le lien, d'assurer les échanges d'informations entre ce centre de traitements et le monde extérieur qu'on réduit, en général, à l'utilisateur.

Le Langage Naturel est un moyen de communication donc d'échange d'informations. Son traitement par la machine se fera donc par l'interface.

On présentera au chapitre 3, les différents objectifs de l'interface et les moyens pour les mener à bien. On montrera que le concept d'interface permet de faire la lumière sur celui de dialogue. On en profitera pour préciser la portée du concept de dialogue de commande.

Ergonomie

Le Langage Naturel est un moyen de communication particulièrement bien adapté à l'homme. Est-il "ergonomique" ?

Il n'est certainement pas du tout adapté à la machine. Pour s'en convaincre, il suffit de comparer la masse de travaux sur le sujet avec le peu de systèmes qui sont actuellement opérationnels.

Son traitement implique que l'on pose un certain nombre de restrictions (Cfr., au chapitre 1, la place du LN finalisé dans notre système d'axes).

Ces restrictions ont-elles une influence sur le caractère ergonomique de l'interaction et, si oui, quels sont les moyens pour minimiser les effets "anti-ergonomiques" ? Ce sujet sera également abordé au chapitre 3.

Recherches sur le multimédia, multimodal [Maybury 92]

La distinction entre interface et application (actions) avait comme but de rendre ces deux entités indépendantes. Les recherches sur le multimédia et le multimodal veulent étendre cette indépendance à l'intérieur même de l'interface.

De plus en plus, on veut que l'utilisateur puisse communiquer avec la machine en utilisant plusieurs modes et moyens de communication.

Comment peut-on étendre les recherches sur le Langage Naturel, notre moyen de communication privilégié, aux autres ? Ont-ils quelque chose en commun ou sont-ils définitivement esclaves de leur nature ?

Si le Langage Naturel est le centre des échange chez l'homme, lorsqu'il communique avec quiconque, il utilise d'autres moyens ou modes (gestes, vue, ...) et il le fait en complète harmonie.

C'est donc que, derrière l'apparente disparité de ces différents moyens de communication, se cachent certains mécanismes fondamentaux identiques.

Le chapitre 6 sera, en partie, consacré à la formalisation de ces mécanismes communs autour du Langage Naturel. Ce sera, encore une fois, l'occasion de montrer que les zones temporelles sont un moyen de représentation unificateur.

Planification

La planification a pour but de déterminer une séquence d'actions pour atteindre un objectif donné.

La conduite d'une application informatique consiste à faire réaliser par la machine les objectifs fixés par l'utilisateur .

Nous verrons au chapitre 3 qu'il est indispensable d'intégrer un planificateur à l'interface, tout particulièrement lorsque l'interaction est en Langage Naturel finalisé, et que ce concept s'adapte facilement, lui aussi, à une représentation des connaissances sous forme de Zones Temporelles (Chapitre 5).

Un projet support

Vu le nombre d'approches qu'on se propose de suivre, il est indispensable de travailler autour d'un cas concret pour garder une certaine cohérence, éviter de se perdre et valider nos recherches.

En septembre 92, Bertrand Gaiffe soutenait sa thèse de doctorat dans laquelle il présentait une implémentation (le projet **Multiworks**) des concepts introduits précédemment par Laurent Romary et qui avait comme but de régler les problèmes de références entre objets de l'application et objets du discours. Ce souci de traiter le mieux possible ces problèmes complexes l'obligea à laisser provisoirement en suspens les questions sur les finalités du discours.

Multiworks est un projet complexe dont un aspect consiste à réaliser une interface qui permet la commande d'une application hyper-texte par le Langage Naturel, dans un premier temps, pour ensuite se tourner vers le multimodal.

Ayant travaillé successivement avec Laurent Romary et Bertrand Gaiffe pendant mon stage au CRIN, notre mémoire s'inscrit donc dans la continuité de ces travaux.

Si, comme le note [Gaiffe 92], *"le caractère naturel ou non du dialogue dépend pour une bonne part des moyens offerts à l'utilisateur pour désigner ces objets"*, nous pensons que l'autre bonne part du caractère naturel ou non d'un dialogue homme-machine dépend de la nature et de la quantité de choses qu'on peut faire faire à la machine avec un énoncé.

Jusqu'à présent, le problème du "contenu en actions" d'un énoncé, pour le projet qui nous occupe (Multiworks), est limité[1] à la relation : un énoncé déclenche une action de l'application. La reconnaissance se limite donc à relier l'action du discours avec celle de la tâche.

S'il on veut concevoir un système de dialogue opérationnel et convivial, cette approche doit être dépassée. Notre travail doit permettre de jeter les fondements d'un système de dialogue qui ne s'intéresserait plus tellement à relier une (ou plusieurs) action(s) de l'application à un énoncé mais plutôt à dégager les finalités des énoncés de l'utilisateur par rapport à la tâche (à quel résultat, état veut arriver l'utilisateur ?) pour ensuite agir de façon optimale. Telle était, en tout cas, notre ambition, en abordant les théories linguistiques des actes de langage.

Il est clair que cette approche apportera d'importants changements à la manière de concevoir l'interface. Jusqu'à présent, elle se contentait de faire le lien entre énoncés et actions (Figure 0.1).

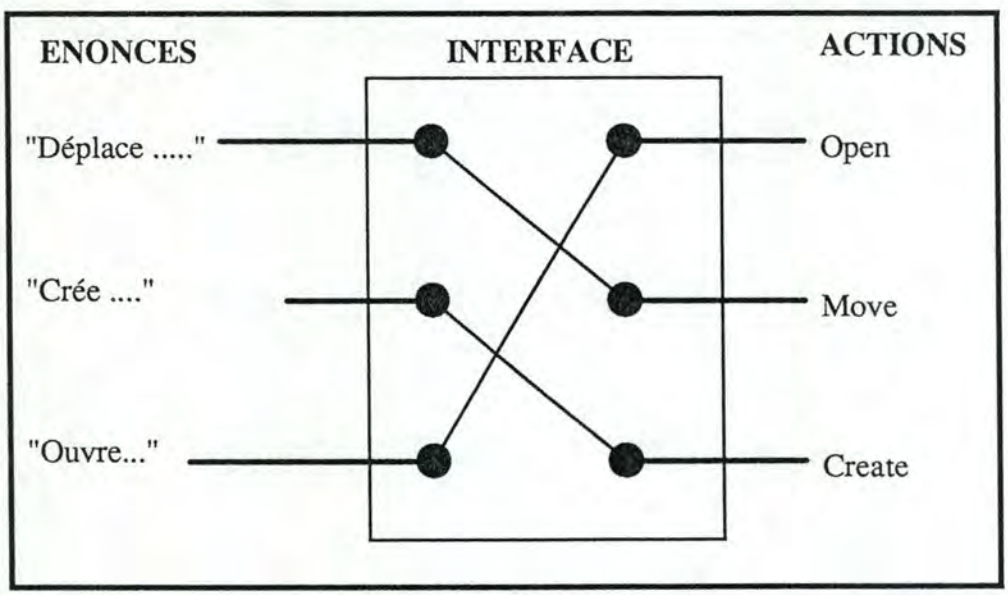


Figure 0.1 : Une interface triviale.

Notre souci est qu'à partir d'une description de ce que veut l'utilisateur, l'interface commande la tâche (Figure 0.2) et ne se contente plus de traduire directement le verbe prédicat de l'énoncé en une action.

[1] Pour que ces problèmes ne perturbent pas inutilement les recherches sur les référents.

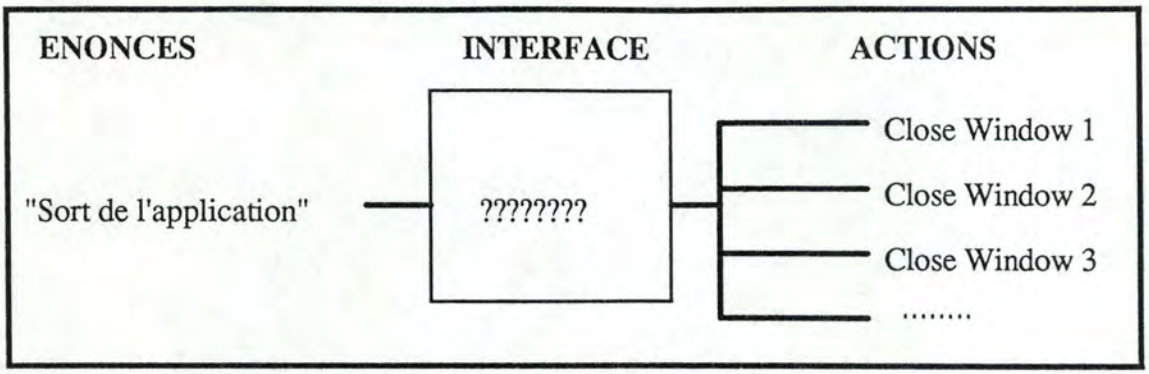


Figure 0.2 Interface évoluée.

En aucun cas, on ne travaillera sur les énoncés directement. On partira en aval du processus de reconnaissance, lorsque le contenu informationnel a déjà été dégagé de l'énoncé.

Tout au long de ce mémoire, on dégagera les composantes nécessaires à l'interface pour mener à bien cette mission. On verra l'importance de la planification dans ce type de problèmes et on en profitera pour recentrer le langage sur sa composante principale : un moyen d'action.

Le second volet de notre travail sera consacré à traduire ces concepts et mécanismes dans le formalisme des Zones Temporelles et à montrer par la même occasion que ce formalisme et la sémantique qu'on lui associe s'adaptent particulièrement bien à ce nouveau type de problème et constitue donc un bon moyen d'intégration des différents types "*d'informations manipulées dans un dialogue homme-machine*". On s'attachera également à entamer une implémentation de ces notions dans le projet Multiworks.

Chapitre 1

1. Le Langage Naturel finalisé, un moyen de communication

Concepts : **communication**, **Langage Naturel (LN)**, **finalité**, **tâche**, **coopération**, **sous-langage**, **Multiworks**.

Ce premier chapitre sera consacré à délimiter la problématique du travail.

Pour ce faire, partant du principe général de communication, nous définirons une typologie du LN et montrerons les difficultés propres à son traitement automatisé. Nous nous servirons de celle-ci pour situer le type de communication qui nous occupe : le pilotage d'une application par le LN finalisé (langage de commande).

La deuxième partie de ce chapitre introductif s'attaquera à présenter l'application et à formuler la question proprement dite : comment peut-on intégrer les concepts de traduction d'intentions en langage, de langage en actes de langage, d'actes de langage en intentions, de planification d'actions, ... à la représentation des actions supportée par **Multiworks** ([Gaiffe 90]), les **Zones Temporelles** (ZT).

1.1. Le Langage Naturel

Partant du principe général de **communication**, nous allons nous focaliser vers un type de communication bien précis, entre un nombre déterminé d'acteurs identifiés, dans une **finalité** imposée. Nous définirons alors une typologie à l'intérieur de ce moyen de communication. Cette restriction progressive de notre espace de travail nous amènera à définir le concept de **sous-langage** introduit par [Deville 89], et à préciser la notion de finalité du discours.

Dans un souci d'étendre les notions au multimédia (modal) (Cfr. chapitre 6), on ne démarrera pas notre travail directement sur des considérations propres au LN mais bien à la communication en général.

1.1.1. Communication Homme-Machine en Langage Naturel finalisé

Procéder par analogie entre l'homme et la machine est une méthode couramment employée en Intelligence Artificielle. Est-elle pour le moins licite ? Pour répondre correctement à cette question, il convient de préciser à quoi correspond cette analogie. Pour qu'un dialogue "marche" entre un homme et une machine il faut que la machine réponde suivant des modes qui sont acceptables par l'utilisateur (ex : conception similaire du temps, de l'espace, ...). Il s'agit d'une analogie de comportement et de conceptualisation. A l'opposé, on peut chercher à copier les mécanismes biologiques (ex : connexionisme). Il s'agit alors d'une analogie de structure. La conception d'un système de dialogue naturel et convivial entre un homme et une machine, passe nécessairement par l'étude des faits et usages de langue chez l'humain. On doit donc procéder par analogie de comportement[2].

Le problème principal de l'homme dans ses relations avec son environnement (et donc les autres hommes) est clairement un problème de communication. Je concède que cette remarque m'est directement inspirée de ma formation d'informaticien. Nous qui n'avons de cesse de faciliter le traitement de l'information, faisons toujours l'hypothèse, ne fût-ce que pour justifier notre travail, que celle-ci circule mal et donc que (sans nous) les hommes rencontrent des difficultés à communiquer.

Le fait que nous utilisions, le plus souvent, le Langage Naturel pour nous exprimer, ne fait qu'accentuer cette difficulté de communication car ce type d'interaction est, par sa nature et par l'ensemble des combinaisons qu'il offre, en continuelle évolution donc non standardisable dans l'absolu, non reproductible en tant que tel, et dès lors, non entièrement compréhensible par tout un chacun; on parle à peu près tous une langue différente[3].

L'homme, en plus de posséder des mécanismes de pensée similaires de par sa nature, jouit d'une incroyable facilité d'adaptation et peut donc se "*mettre au niveau* "de son interlocuteur et comprendre ce qu'on lui veut. On voit tout de suite les difficultés que rencontrera une machine dans cet exercice ou plus précisément, les difficultés que va rencontrer le programmeur de cette machine pour lui permettre cet exercice.

Un modèle de la Communication, le modèle de Shannon

Dans ce mémoire, nous ne nous intéresserons que très peu au fait que le langage d'interaction est avant tout le LN. Nous ne travaillerons pas sur le signal de langage directement mais plutôt sur son contenu informationnel. Dans un souci de généralité, il était primordial de commencer ce travail par la présentation d'un modèle de la communication accepté par tous qui ne se limitait pas au LN (Figure 1.1).

Examinons le modèle de Shannon [Deville 89].

---

[2] Ces réflexions me sont directement inspirées de remarques de Laurent Romary.

[3] Ne fût-ce que par la fréquence de vibrations de nos cordes vocales.

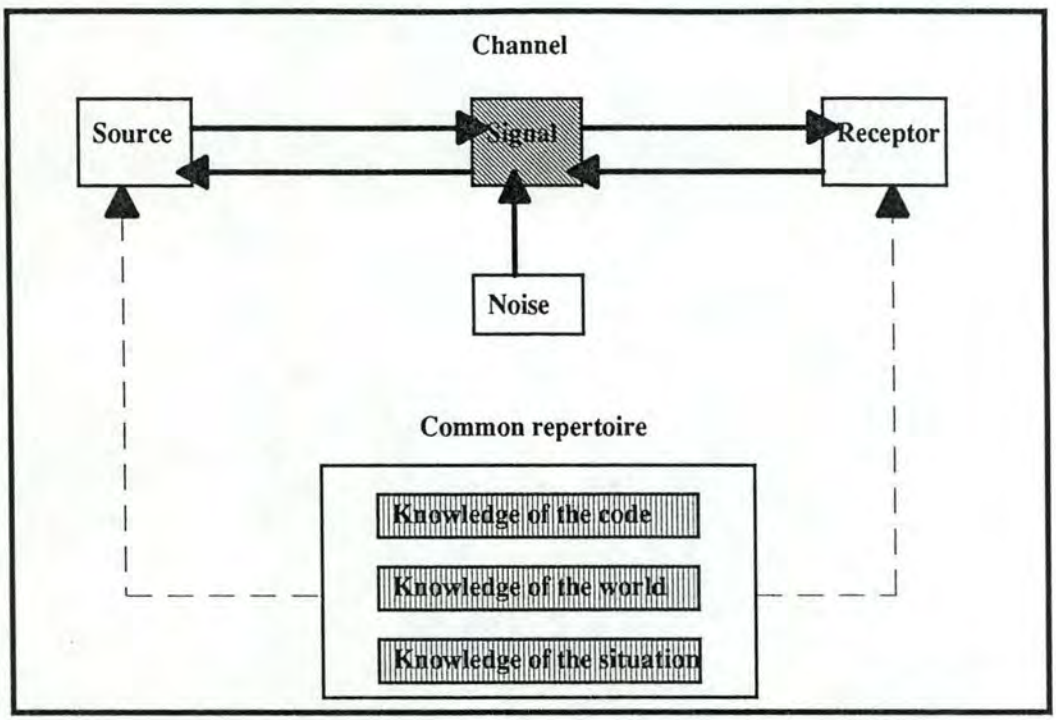


Figure 1.1 : A general model of communication

La communication est un processus qui implique au minimum deux acteurs, qui sont tour à tour émetteur et récepteur d'un signal. La transmission de ce signal se fait sur un canal de communication et peut être altérée par du bruit en fonction de la nature du canal.

La communication entre ces deux acteurs nécessite un certain nombre de connaissances communes :

- Knowledge of the code (connaissance du code) : des connaissances sur la façon dont on code ce que l'on veut communiquer en un signal.

- Knowledge of the world (connaissance du monde) : des connaissances sur le sujet de la communication (pour communiquer il faut avant tout avoir quelque chose à communiquer !).

- Knowledge of the situation (connaissance de la situation) : des connaissances sur l'état de la communication (savoir ou on en est dans ses échanges).

Précisons en quoi l'utilisation du LN Homme-Machine Finalisé comme moyen de communication, instancie et restreint les concepts génériques du modèle de la figure 1.1.

Communication en LN

En introduisant la notion de LN à notre modèle (figure 1.1), on définit une première restriction à la généralité; une restriction sur le canal de communication (channel).

On limite par la même occasion le système de codage que l'on va utiliser ainsi que les connaissances sur le code (Knowledge of the code).

Dans l'absolu, il suffit de disposer de cet ensemble de règles de codage et de permettre le branchement au channel adapté au LN pour pouvoir communiquer.

Mais qu'entend on exactement par LN [4]? Le LN est avant tout défini par l'ensemble des règles lexicales, syntaxiques, sémantiques, ... des différentes langues (ajoutons les règles phonétiques, phonologiques, ... lorsque le langage est parlé et non écrit). Mais nous pensons que le LN ne se limite pas aux règles de grammaire usuelles. Le LN est avant tout un moyen d'expression vivant donc dynamique, hyper flexible et pas du tout formalisable. Son traitement (émission, reconnaissance, ...) demande à l'homme d'avoir un esprit ouvert.

Hélas, si l'homme est capable de s'adapter aux autres, d'adapter son système de codage en fonction de la personne à laquelle il fait face et tout cela sans s'en rendre compte, il lui est beaucoup plus difficile d'expliquer a posteriori comment il y est arrivé.

Arriver à circonscrire l'ensemble des règles de codage-décodage de pensées en LN pour, ensuite, appliquer ces concepts formels stricts à une matière aussi dense que le langage reste l'objectif ultime mais improbable de toute recherche sur le LN.

Communication Homme-Machine

Ici, l'instanciation de la figure 1.1 touche les acteurs de la communication : un homme et une machine.

Comme je l'ai déjà dit, l'homme est un être doué d'une facilité d'adaptation à son environnement.

La tendance première de l'informatique s'est satisfaite de cette constatation. Pendant pas mal d'années, l'homme s'est astreint à communiquer avec la machine en employant un signal le plus proche possible de la façon dont cette machine gérait ses connaissances (cartes perforées, assembleur, ...). Le processus de codage-décodage est resté pendant longtemps à la charge de l'homme.

Si l'homme, en temps que système auto-régulé, s'adapte très vite, il est également doté d'un certain pouvoir d'abstraction, de génération de solutions, et d'une fameuse dose de paresse qui le poussent à inventer des solutions qui rendent son travail de moins en moins pénible. C'est ainsi que sont nés les premiers langages formels de programmation et la volonté chronique d'introduire des systèmes intermédiaires de traduction du signal entre l'homme s'exprimant en LN et la machine ne comprenant que du binaire; jusqu'aux systèmes de communication actuels.

---

[4] Cette question philosophique n'est pas traitée dans notre travail. On se contentera de définir plus précisément ce qu'est un langage de commande.

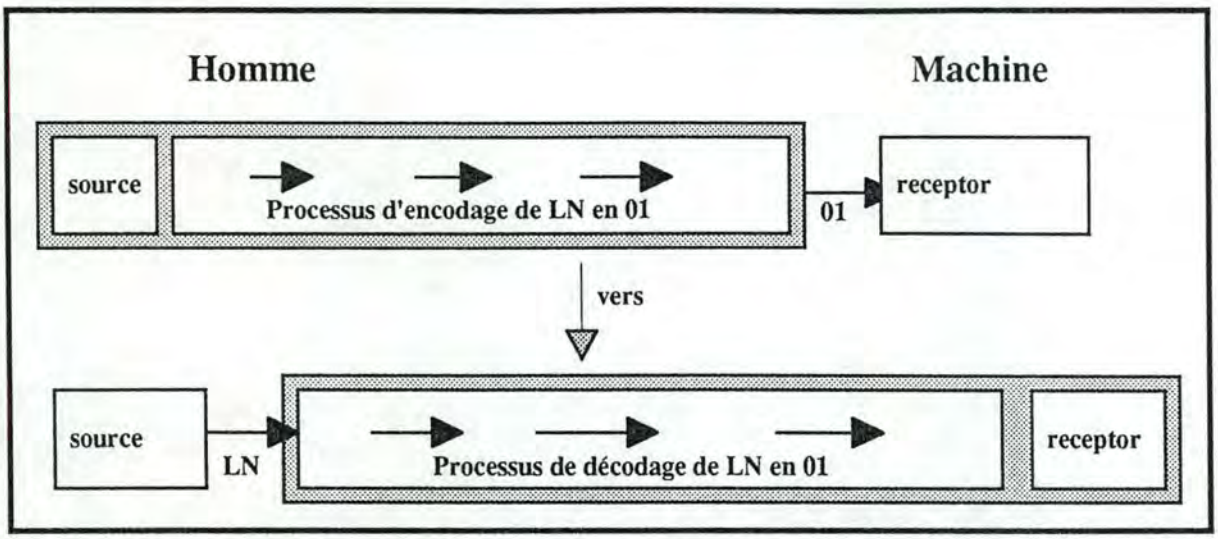


Figure 1.2 : Déplacement du codage-décodage de l'homme à la machine.

La figure 1.2 exprime le tendance à faire glisser le processus de codage-décodage de l'homme vers la machine et doit sa simplicité à au moins deux prises de position de notre part.

Premièrement, elle aurait tendance à laisser croire que les croyances de l'homme ont une représentation interne en Langage Naturel. Ce n'est évidemment pas le cas; par introspection, on se rend bien compte qu'on ne pense pas qu'en termes de LN, mais également en couleurs, images, odeurs, et que sais-je encore.

Deuxièmement, dans une communication traditionnelle, chaque acteur est tour à tour émetteur et récepteur de signal. Dans le cadre du dialogue Homme-Machine, nous nous sommes contentés d'attribuer le rôle d'émetteur à l'homme et celui de récepteur à la machine; il est clair que cela constitue une réduction à la généralité du modèle, mais se rapproche plus de la pratique (notre pratique). Si la machine veut communiquer un signal à l'homme, elle le fera le plus souvent (dans le type d'application auquel s'intéresse notre travail) par un autre type d'interaction que le LN (en affichant quelque chose à l'écran par exemple).[5]

Ceci dit, on constate que depuis le début de l'informatique, l'homme n'a eu de cesse d'intégrer le processus de codage-décodage à la machine et de se soulager ainsi de ce travail.

On a présenté ici deux extrêmes[6]; il est clair que dans les systèmes actuels, le processus de codage-décodage est partagé.

Communication Finalisée

Lorsque la communication est finalisée, on restreint encore l'étendue de la figure 1.1.

---

[5] Dans le projet Multiworks, comme défini plus loin, la problématique est de définir un système de traitement du LN par la machine pour que celle-ci comprenne et exécute les ordres qui lui sont transmis par l'utilisateur. La machine, dans l'absolu et en supposant que l'interprétation de ces ordres ne pose pas de problèmes, ne communiquera avec l'utilisateur qu'en présentant les résultats de l'exécution de ceux-ci.

[6] Communication en binaire versus en LN.

Finalisé est-il synonyme de limité? Nous ne le pensons pas.

Par **communication finalisée**, on entend une communication qui se fait dans un but précis, autre que celui d'émettre simplement un signal. La finalité porte donc sur un certain but, une certaine tâche [7] à exécuter.

Si la communication n'est a priori pas limitée, la tâche l'est bien. La limitation sous jacente à la finalité porte donc sur ce que nous avons appelé, à la figure 1.1, la "*knowledge of the world*" .

Nous verrons par la suite lorsque nous aborderons le concept de sous-langage en quoi cette limitation de l'univers du discours limite le discours lui-même.

1.1.2. Où nous situons-nous dans le Langage Naturel ?

Maintenant qu'on a réduit la communication à une communication finalisée en LN entre un homme et une machine, centrons-nous sur le LN proprement dit.

Tentons, d'une part, d'introduire une classification du LN pour mieux situer notre propos (Multiworks); voyons, d'autre part, comment on peut se contenter de la notion de sous-langage lorsque le LN est Finalisé.

Un système d'axes

Reprenons la typologie introduite par [Mousel 90], en y apportant quelques remarques, modifications et ajouts. Cette typologie se présente sous la forme de systèmes d'axes dialectiques.

La finalité de cette typologie est clairement de "**limiter le problème**".

La complexité du langage

La complexité du langage est due au double fait que, d'une part, il existe un nombre indéfinissable de façon d'arranger un signal pour faire passer une idée; et que d'autre part, un même signal (nous reprenons les concepts de Shannon Figure 1.1) peut représenter, à lui seul, plusieurs idées.

[7] Pour une définition du concept de **tâche**, on renvoie le lecteur au chapitre 2.

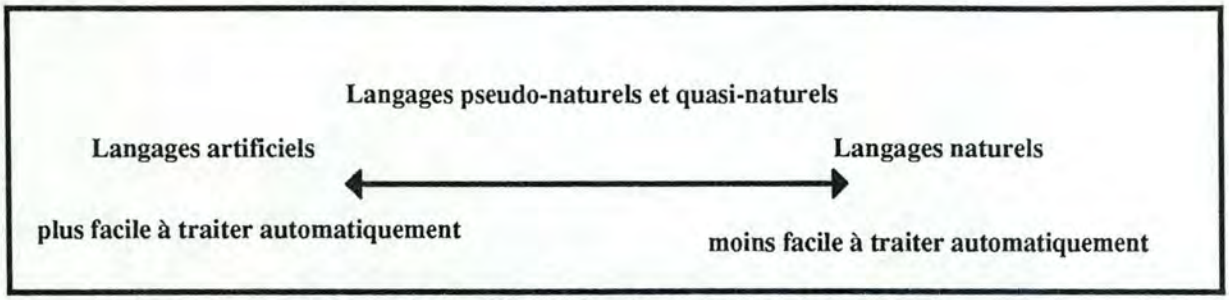


Figure 1.3 : La dimension "complexité du langage".

Dans la figure 1.3, on retrouve aux extrêmes les langages artificiels (parfaite bijection entre le signifiant et le signifié) et les langages naturels; entre quoi, on a défini un certain nombre de langages intermédiaires dans un but évident de faciliter le traitement (la compréhension automatisée) des signaux produits. Cette tendance a déjà été évoquée à la figure 1.2.

Si les langages artificiels ont des règles de construction simples (lexique limité, syntaxe souvent de la forme : prédicat + compléments, ...) et précisent des idées claires, ils n'en sont pas moins fort loin de la façon dont tout un chacun s'exprime, pas souples, contraignants, ..., bref, pas "naturels" du tout.

Pour le projet qui nous occupe, on ne se donne aucune limitation sur le discours a priori.[8]

La complexité de l'univers du langage

Alors que le premier axe traitait de la complexité du langage lui-même, le deuxième s'attache à classifier les sujets du discours.

Tout comme il est clair qu'on ne s'adresse pas à un bébé de la même façon qu'à un adulte (langage différent), on se voit mal débattre de politique ou de philosophie avec un bambin (sujet différent).

Il faut adapter le sujet de son discours à la personne à qui on s'adresse. Un ordinateur est une machine. Une machine existe pour effectuer des tâches. On va donc s'adresser à elle pour qu'elle réalise ces tâches et uniquement en se mouvant dans l'univers langagier des tâches qui nous occupent.

L'univers du langage sera d'autant plus réduit que ces tâches sont triviales [9].

[8] C'est actuellement faux, les rares commandes à avoir été implémentées pour les besoins des différentes démonstrations sont toutes de la forme Prédicat + complément. (Néanmoins, le but est bien d'arriver à une liberté maximale dans la forme du discours.)

[9] Cfr. les univers de blocs modélisés par Winograd en 1972 par exemple (SHRDLU) in [Charniak 76].

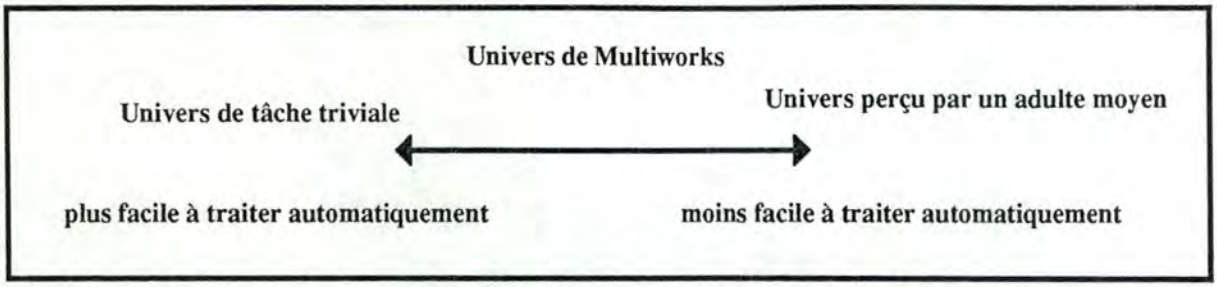


Figure 1.4 : La dimension "complexité de l'univers du discours".

Plus l'univers est réduit et plus le traitement automatique des "accès" à cet univers est facile.

L'univers de Multiworks est un univers d'interface d'hyper-document constitué d'éléments et d'opérations sur ceux-ci en nombre limité mais loin d'être simpliste (nombre important de combinaisons, ...). On l'a donc placé plus ou moins au centre gauche de la figure 1.4[10].

L'univers (concept utilisé par [Mousel 90]) est, en fait, l'ensemble des objets et des actions de la tâche commandée par un utilisateur. On préférera, dans les chapitres qui suivent, parler de **modèle de la tâche**.

L'initiative

Savoir qui a l'initiative revient à se demander quel acteur sera maître du discours (et du sujet du discours) et pourra donc prévoir, anticiper, faciliter le dialogue. *"L'initiative fait référence au fait que c'est l'utilisateur ou bien l'ordinateur qui dirige les transactions au sein du dialogue"* [Scapin].

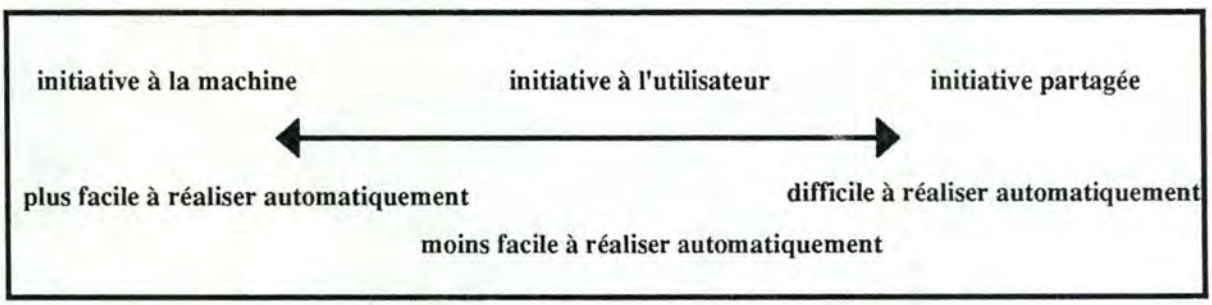


Figure 1.5 : La forme de l'interaction.

Le système classique d'interfaçage est le menu (plus ou moins complexe et plus ou moins explicite). Par ce type d'interaction, bien que ce soit l'utilisateur qui sélectionne l'opération qu'il veut exécuter, nous pensons que l'initiative est entièrement à la machine. C'est elle, en effet, qui d'une certaine manière débute l'interaction en offrant ses services. Ce mode de fonctionnement est actuellement assez aisé à automatiser et très répandu.

A l'opposé, on peut mettre les langages de commande, comme celui supporté par Multiworks. Selon nous, ce type d'interaction est à l'initiative de l'utilisateur. Il ne se contente

[10] Il est difficile de placer un élément sur une échelle aussi floue. C'est le danger de toute abstraction.

plus ici de sélectionner une action dans une liste présentée par la machine, il définit lui-même si pas l'action, du moins la séquence de leurs activations, limitant ainsi le pouvoir d'anticipation de la machine.

Entre les deux, on retrouve l'initiative partagée. Un système de communication est à initiative partagée lorsque n'importe quel acteur peut changer le sujet de la conversation. La difficulté de gérer ce type de communication est énorme car dans ce cas, presque aucune anticipation n'est possible.

Les systèmes à initiatives partagées entre un homme et une machine sont des systèmes fort complexes, les plus intéressants, mais encore trop contraignants : on est obligé de limiter le discours et/ou l'univers du discours de façon drastique pour rendre ces systèmes opérationnels[11].

Reconnaître versus Comprendre

La dernière dualité soulignée par la typologie s'intéresse au fait de savoir s'il faut privilégier la forme ou le fond pour traiter automatiquement le LN.

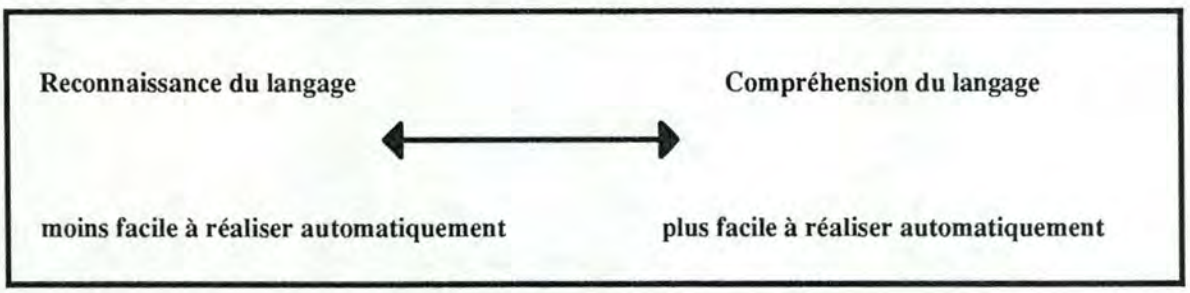


Figure 1.6 : La dimension de la "finalité du traitement".

La reconnaissance du langage oral (ou reconnaissance de la parole) porte son attention sur la forme; les systèmes les plus représentatifs de ce mode de traitement sont les machines à dicter.

La compréhension quant à elle s'intéresse aux idées, intentions traduites en langage.

Il est clair que pour mieux traiter la forme, il faut s'intéresser au fond (ne fût-ce que pour résoudre les problèmes d'homophonie). L'inverse est d'autant plus vrai.

Pour notre part, nous situerons notre travail à la fin du processus de traitement du signal et nous ne nous intéresserons que fort peu à la forme; nous considérerons que toute l'information interne à la structure du signal aura déjà été analysée et prise en compte. Cette décision est motivée par plusieurs arguments.

Le premier est que pour des raisons évidentes de temps et de division du travail, il est impossible de développer un système complet de reconnaissance de la parole dès que l'on veut

[11] On peut citer, par exemple, le système de renseignements téléphoniques développé par l'équipe dialogue du CRIN [Roussanaly 88], [Mousel 90].

s'attaquer à un problème précis (pour Multiworks, ce système de reconnaissance a d'ailleurs été développé à Grenoble).

Ensuite, cela nous permettra de faire abstraction de tous les problèmes liés au traitement du Langage Naturel lui-même pour nous concentrer sur la réalisation de nos objectifs proprement dits.

Enfin, en ne s'attachant pas à un type de communication particulier, on garde l'esprit ouvert au multimédia (modal) en essayant de généraliser les travaux spécifiques aux actes de langage à d'autres types d'interactions (cfr. chapitre 6).

Le projet Multiworks est un projet du type compréhension du LN. Le LN ne sert qu'à décrire comment la tâche doit évoluer et n'est donc jamais une fin en soi.

Les sous-langages[12]

La présente typologie nous amène à faire des choix (sur la complexité du langage, des sujets, de la structure des échanges et du niveau de compréhension).

Ces choix bien que présentés de façon indépendante dans des paragraphes séparés ne le sont en rien.

En introduisant le concept de sous-langage, [Deville 89] illustre clairement en quoi la limitation des finalités du langage transpire sur la limitation du langage lui-même.

Présentation du concept

Un sous-langage est "*un ensemble d'énoncés faisant référence à un domaine sémantique limité et bien défini et utilisé pour une fonction spécifique. De tels énoncés sont engendrés par une grammaire spécifique et un vocabulaire spécifique*".

La démarche qui a précédé cette définition partait des principes suivant :

- Le langage n'est a priori pas limité.
- L'univers du discours, lui est limité; on a affaire à un langage finalisé.

Si on considère que le langage est constitué d'éléments, de composants lexicaux, syntaxiques, sémantiques et pragmatiques (pour des définitions précises de ces concepts et de leurs découpes, on peut se référer à [Pierrel 87], [Sabah 89]) et que la limitation de l'univers du discours limite le lexique, la syntaxe, la sémantique et la pragmatique; on doit se rendre à l'évidence que c'est le langage qu'on limite (on définit donc de façon implicite et probabiliste un sous domaine du LN).

---

[12] Le concept de sous-langage sommairement présenté, par son caractère archi-général, est plus qu'insuffisant pour décrire ce qui se passe dans un dialogue Homme-Machine. Il est néanmoins pertinent pour expliquer les liens entre connaissances sur l'univers du discours et connaissances du code, et plus précisément pour mettre en évidence qu'une limitation du premier entraîne inévitablement une restriction sur le second.

Les sous-langages ont, par leur aspect très orienté vers la tâche, la propriété d'accepter en plus des formes lexicales, syntaxiques, sémantiques qui seraient non pertinentes dans le langage traditionnel (dans le sens repris par les linguistes).

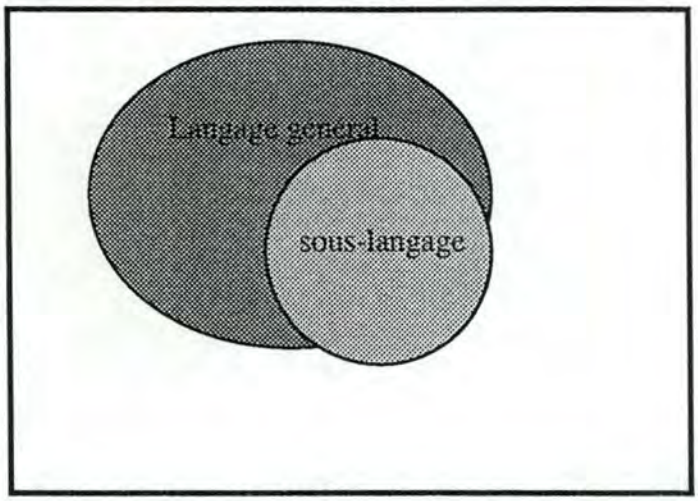


Figure 1.7 : Place des sous-langages dans le langage en général.

Un sous-langage est donc inférieur au langage général mais il peut être également en partie en dehors (figure 1.7).

Toute limitation impose un certain respect de principes, une certaine coopération des acteurs. Ceci nous amène à définir des hypothèses de coopérations, pré-conditions à l'utilisation licite de tout sous-langage.

Les maximes de Grice

Dans sa théorie de l'implication, Grice tente de définir une ligne de conduite pour une utilisation du langage qui soit opérationnelle et efficace dans une conversation (communication en LN) ([Levinson 83]).

Il définit un principe à ne jamais transgresser :

• **le principe de coopération** : "*make your contribution such as required, at the stage at which it occurs, by the accepted purpose or direction of the talk exchange in which your are engaged*".

Il définit ensuite des maximes qui assurent une conduite coopérative de la conversation . Sans les définir toutes, citons : les **maximes de quantité** ("*make your contribution as informative as required for the current purpose of the exchange*", ...), **de qualité** et **de manière**, ...

Dans une conduite normale de la conversation, les maximes et le principe sont respectés.

Toute infraction à une maxime entraîne a priori le rejet du principe de coopération; or ce principe est inviolable si le dialogue se veut coopératif.

C'est donc que pour rester coopératif, on a dû violer une maxime, introduire de ce fait une rupture dans la séquence de la conversation et attirer l'attention de la partie adverse sur un élément sous-entendu par le non-respect de cette maxime. On explique de la sorte comment un énoncé peut être plein de sous-entendus.

L'aspect "coopération" entre utilisateur et machine dans la conduite d'une application est primordial.

Ainsi, [Souvay 92] définit un dialogue finalisé ou dialogue de commande comme un "*dialogue coopératif orienté par la tâche*"

Lorsqu'on présentera les actes de langage (Cfr. chapitre 2) et qu'on parlera de la validation de l'interprétation des énoncés par la planification (Cfr. chapitre 3, 5), on verra en quoi le respect du principe de Grice est fondamental et doit guider nos choix d'implémentation (notamment, l'obligation d'informer l'utilisateur des modifications de la tâche, Cfr. chapitre 6).

1.1.3. Le langage de commande, un sous-langage

En situant le langage de commande (Langage Naturel coopératif finalisé) dans notre système d'axes, on définit clairement un sous-langage (figure 1.8).

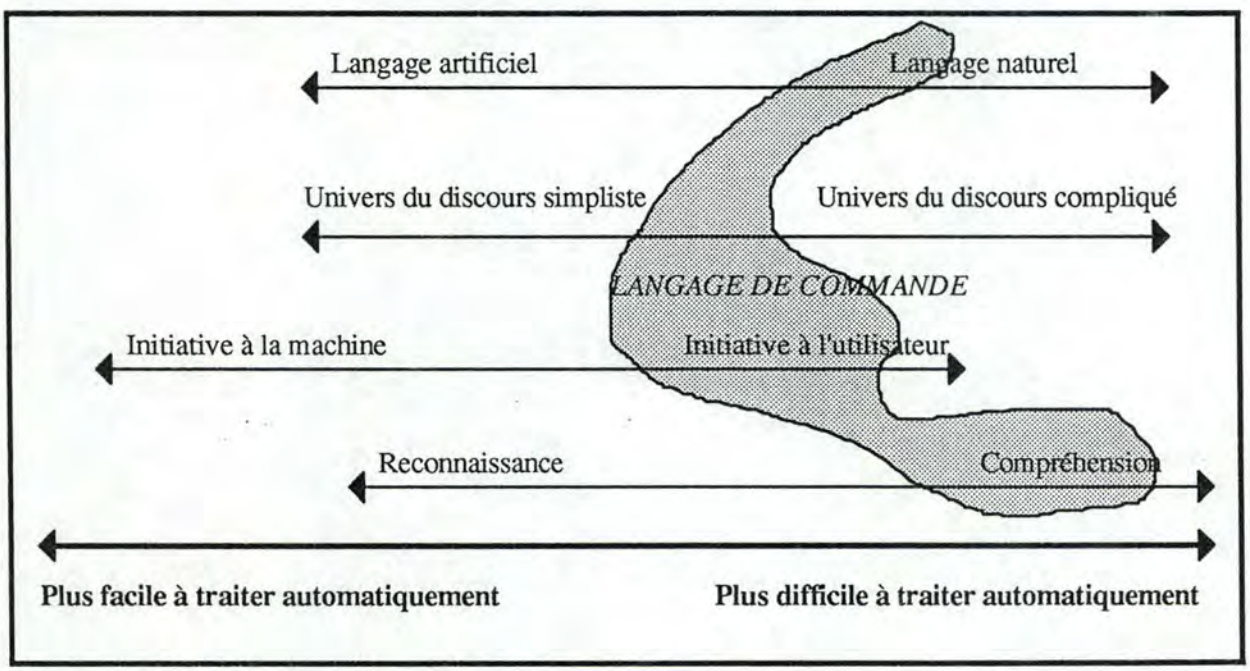


Figure 1.8 : Le langage de commande dans notre système d'axes.

Le **langage de commande** est un langage finalisé; il se rapproche donc du pôle d'univers du discours, si pas simpliste, du moins entièrement formalisé. Il est à l'initiative de l'utilisateur et s'attache principalement à la compréhension, au décodage par la machine de ce qu'il a bien pu dire.

1.2. L'application

1.2.1. L'existant

Multiworks est un projet Esprit (ESP 2105) dont l'objectif était de développer une station de travail intelligente. Comme tout projet européen, il concernait nombre de chercheurs dans plusieurs de pays de la communauté. La partie supportée par l'équipe dialogue du CRIN, consistait à définir et développer une maquette d'interface multimodale d'une application de gestion d'hyper-document : MULTICARD. Elle se centrait sur trois optiques :

- Le dialogue est finalisé : gestion d'un dialogue entre le concepteur d'un document et l'éditeur hyper-texte MULTICARD.

- Dialogue de commande : on se limitera à traiter des énoncés de la forme prédicats à l'impératif + Arguments.

- Dialogue multimodal : possibilité de désigner des objets à la souris et bientôt à l'aide d'un gant de désignation.

Multiworks est développé autour du concept de Zones Temporelles [Romary 89]. Cette représentation des connaissances à été développée jusqu'ici pour faire face aux problèmes de calcul de références (traitement des anaphores, ellipses, ...) entre les énoncés et les objets d'une tâche. Multiworks a servi de prototype à l'implémentation de ce concept et de ce qui en découle [Gaiffe 92].

Tout énoncé devant être interprété en situation pour résoudre les problèmes de référence, les Zones Temporelles paraissaient particulièrement bien adaptées pour modéliser la tâche.

Jusqu'à présent, on se contentait d'exécuter directement des actions "simples", réduisant ainsi le dialogue à sa plus simple expression. L'objectif était de vérifier que les objets modifiés par ces actions simplistes avaient été correctement identifiés. S'il on veut étoffer les actions reconnues et donc le dialogue, on se doit d'incorporer au système une modèlisation étendue de la réalité et un module de planification.

1.2.2. La tâche : MULTICARD

Typiquement, un document hyper-texte est constitué d'objets hiérarchisés et admet un certain nombre d'opérations sur ceux-ci.

A un niveau purement logique, au sommet de la hiérarchie, on trouve l'objet groupe. Un groupe est composé d'un ensembles de noeuds reliés entre eux par des liens.

Chaque noeud est constitué d'un certain nombre d'ancres auxquelles est associé un script. Lorsqu'on active l'ancre (par exemple en cliquant sur un bouton), on déclenche le script qui lui est associé (par exemple, on entend un chant d'oiseau).

A côté de cette structure logique, on retrouve des objets permettant la visualisation de ces derniers à l'écran : fenêtres, icônes, boutons, ... Les opérations sur les objets logiques (créer un noeud dans un groupe, créer un lien, ...) cohabitent avec celles sur les objets physiquement représentés à l'écran (figure 1.9). Par exemple, en cliquant sur un bouton, on ouvre un noeud .

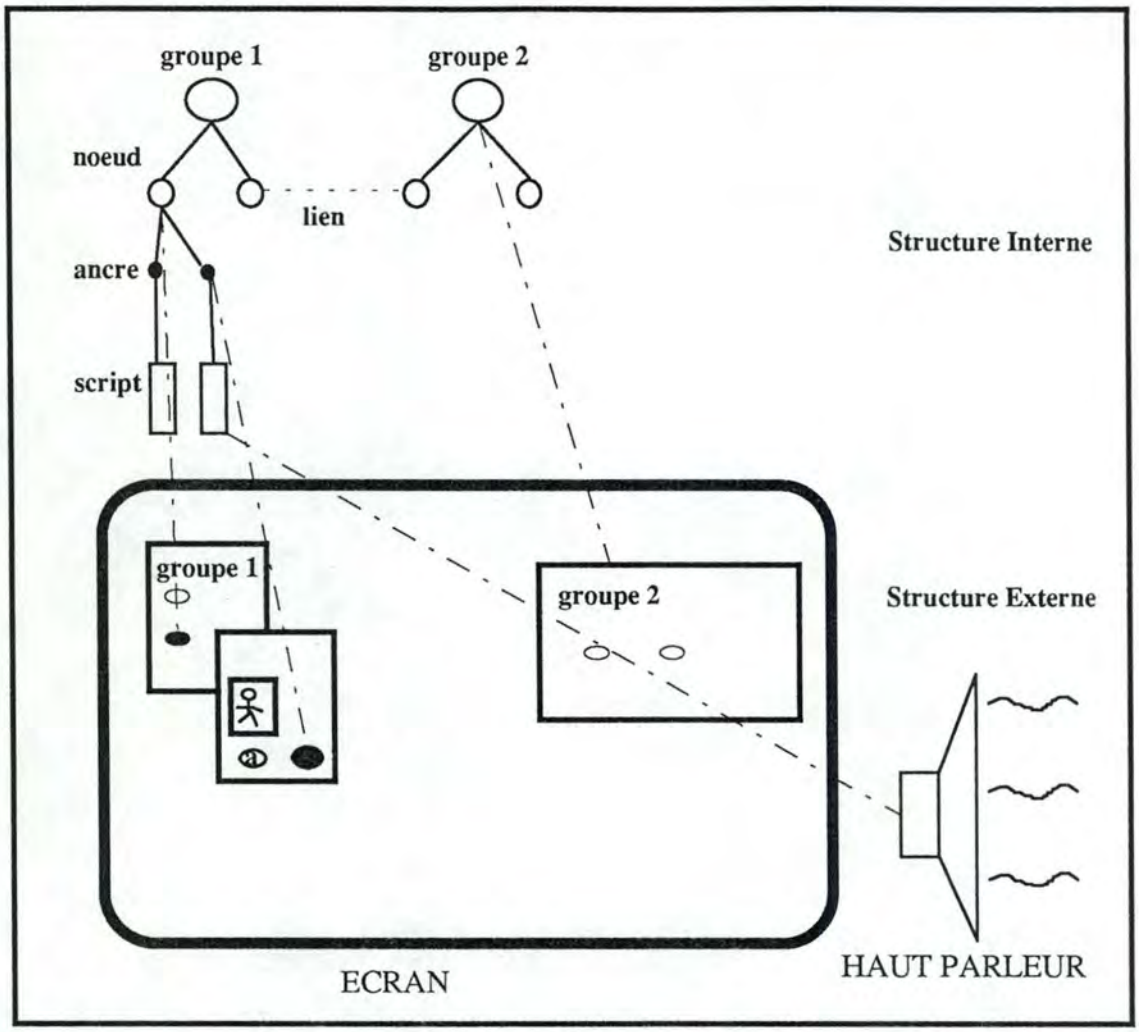


Figure 1.9

Ce type d'architecture de document permet de nombreuses possibilités référentielles et oblige l'interface à se doter d'une représentation des objets souple, évolutive et assez simple.

Il était normal, dés lors, que l'on se tourne vers un mode de représentation temporel.

1.2.3. Développement

On se donne comme objectifs de traduire les notions d'intentions, de transmission d'intentions, d'actes de langage (chapitre 2 et 3), de planification et d'actions génériques en zones temporelles et donc d'affiner le modèle (chapitre 4), en sorte d'augmenter les capacités de dialogue de l'interface et d'y intégrer d'autres modes que le mode langagier.

On développera un planificateur dont le but sera bien sûr de trouver (le plus vite) et le mieux possible une séquence d'actions de la tâche qui réalise la requête mais qui retourne également les informations les plus pertinentes sur l'échec de la planification en cas de requête non réalisable. Ce planificateur servira de support à une meilleure interaction entre l'utilisateur et la machine (chapitre 5).

On essayera également de définir un processus d'agrégation des actions (processus d'apprentissage) qui permettra de soulager la planification et servira dans le futur de support à la production de réponse indépendamment du mode de transmission choisi (chapitre 6).

1.3. Conclusion

Toute communication nécessite des connaissances communes des acteurs sur le code, le monde et la situation (Figure 1.1).

La communication entre humains fonctionne correctement car les acteurs sont plus ou moins identiques au départ (les hommes, au même titre qu'ils ont deux bras et deux jambes, possèdent des mécanismes de pensée qui sont intrinsèquement similaires, même si l'expérience de chacun est différente) et jouissent de facilités d'adaptation surprenantes.

Ce n'est plus le cas lorsqu'un des interlocuteurs est une machine. On choisira alors de faire en sorte que la machine se conduise comme un homme.

Le LN est le centre de la communication pour l'homme. Si on limite le LN à un Langage coopératif de commande d'un homme vers une machine sur une tâche précise, on limite le problème, mais l'objectif reste fondamentalement le même : donner à la machine des connaissances sur le code (on s'y intéressera fort peu par la suite), le monde et la situation; et des mécanismes d'adaptation et de régulation de ces connaissances analogues à l'homme.

Chapitre 2

2. Le Langage Naturel Finalisé, un moyen d'action

Concepts : **action, état de connaissance réel et intentionnel, acte de langage, but.**

Au cours de ce chapitre, on s'écartera du projet Multiworks : les concepts introduits étant, en principe, indépendants de l'application. Le but est de voir ce qui est propre à l'utilisation du LN finalisé comme moyen de commande quelque soit la tâche commandée, pour ensuite mieux y retourner.

Ce chapitre, tout comme le suivant, servira à définir les concepts que l'on veut intégrer à Multiworks, dans une syntaxe, représentation qui n'est pas celle de Multiworks. Nous n'introduirons les zones temporelles qu'au chapitre 4. Nous croyons qu'il est préférable de s'attarder sur les concepts centraux de notre travail en les présentant de la façon la plus naturelle (la plus fréquemment utilisée dans la littérature) et ensuite seulement, adapter ceux-ci à notre représentation temporelle.

Nous articulerons notre réflexion autour d'une logique d'actions basée sur l'hypothèse suivante : le seul moyen d'évolution d'un système est l'action.

Après avoir défini quelques concepts fondamentaux, nous montrerons que le langage permet la transmission d'intentions entre un consommateur et un producteur de service (une tâche).

Cette transmission d'intentions trouve sa validation dans notre logique lorsqu'on introduit le concept d'acte de langage contenu dans tout énoncé. Nous résumerons donc la théorie des **actes de langage** pour n'en retenir que les actes principaux rencontrés dans les langages de commande (request, inform, question).

2.1. Définitions

Avant d'aller plus loin, il est peut-être bon de donner des définitions aux concepts et de poser les principes qui seront les nôtres tout au long de ce mémoire.

La **réalité** est l'ensemble des choses qui ont une existence au temps T. Nous intéressant au langage de commandes finalisé entre un homme et une machine, nous limiterons la réalité à ces deux acteurs et plus particulièrement à l'application supportée par la machine.

Une **application** est "*un traitement quasi-autonome par rapport aux autres applications d'un projet : elle constitue une unité de planning dans la gestion d'un projet...*" [Bodart 89]. C'est en quelque sorte l'entité qui réalise la tâche [13].

Chacun perçoit la réalité (Cfr. Mythe de la caverne de Platon), se fait une représentation interne de celle-ci, il faut l'espérer, la plus bijective possible.

On appelle cette représentation, **monde réel** ou encore état réel de représentation ([Mignot 92]). Dans un premier temps, on limitera le monde réel à une modélisation de la tâche réalisée par l'application.

Une **tâche** est, d'un point de vue statique, un ensemble d'éléments (des objets) et d'opérations (des actions)[14] couvrant un domaine cohérent, une spécialité.

Une **action** (acte, opération, ...) est un processus qui est capable de faire évoluer un ou plusieurs **objets** (ou actions, on parlera alors de méta action) donc la tâche. Le seul moyen de faire évoluer une tâche (objets ou actions) est de lui appliquer des actions.

Un **acteur** est une entité qui modifie une tâche, qui agit directement dessus.

---

[13] Doit-on y inclure l'interface ?

[14] Le concept d'objet est proche de celui généralement repris par les méthodes "**orientées objets**" (un objet est une entité autonome, constituée d'attributs et d'actions internes modifiant ceux-ci; ces actions pouvant déclencher également des actions dans un autre objet par l'envoi de messages; ...)

En ce qui nous concerne, nous considérerons qu'un objet est une entité autonome représentée par l'ensemble de ses attributs. A côté de ces objets, nous définirons un certain nombre d'actions externes : des actions qui s'appliquent à certains objets, qui ont donc des objets comme arguments.

Un **état de la tâche** (modèle de la tâche, de l'application, ...) est une représentation de la tâche à un moment donné et plus précisément des objets de la tâche si on considère que les actions sont invariantes.

Un monde intentionnel est un ensemble d'intentions. Une intention est une représentation non encore valide, un voeux sur la réalité. De la même façon, si on limite la réalité à la tâche, une intention sera une représentation anticipée de celle-ci.

Un utilisateur de la tâche est est une entité qui a des intentions sur celle-ci

2.2. Les intentions dans le langage

Comme nous venons de le dire, seule une action est capable de modifier la tâche. Ces modifications, pour faciliter notre travail, seront considérées comme directement portées à la connaissance des entités en contact avec elle. Le monde réel de chacun sera donc instantanément modifié.

Comme seuls les acteurs (modules de l'application) sont capables d'exécuter une action, l'utilisateur ne saura pas agir directement sur la réalité (et modifier ainsi son monde réel); il devra passer par une logique intentionnelle et le langage de commande.

Pour illustrer notre propos, supposons que je rêve de déguster un gâteau au chocolat. Faisant partie d'une société où le mode de production est basé sur la division du travail, je suis un informaticien de grande classe mais (et donc) incapable de produire autre chose que de l'informatique. Je suis donc obligé de me rendre chez un spécialiste : un pâtissier.

2.2.1. Traduction des intentions en langage

Ne sachant pas subvenir à mes besoins (intentions), il va falloir que quelqu'un d'autre y pourvoit. Pour ce faire, je vais devoir traduire mes besoins en langage en essayant d'être le plus bijectif possible. *"Bonjour, je voudrais un gâteau au chocolat s.v.p."*, me semble une bonne traduction de mes intentions en LN.

Savoir comment j'y suis arrivé est un problème qui ne nous occupe pas pour l'instant.

2.2.2. Traduction du langage en intentions

Recevant mon énoncé, le pâtissier va en retirer que je veux un gâteau, que comme il n'existe pas et que je le lui demande, il doit me le fabriquer, ... il va donc modifier son monde intentionnel de telle sorte que la production d'un gâteau deviennent pour lui une intention.

Comme j'ai supposé que, pour qu'un monde (réel ou intentionnel) évolue, il faut poser une action, c'est donc que l'énoncé *"Bonjour, je voudrais un gâteau au chocolat s.v.p."* en contenait.

Mon intention de me procurer un bien devient donc une intention pour le pâtissier de me le fournir. Comme il est en mesure de transformer cette intention en réel, il s'exécute, me donne mon gâteau et de ce fait, rétablit l'équilibre entre nos mondes intentionnels et réels.

2.2.3. Un modèle synthétique

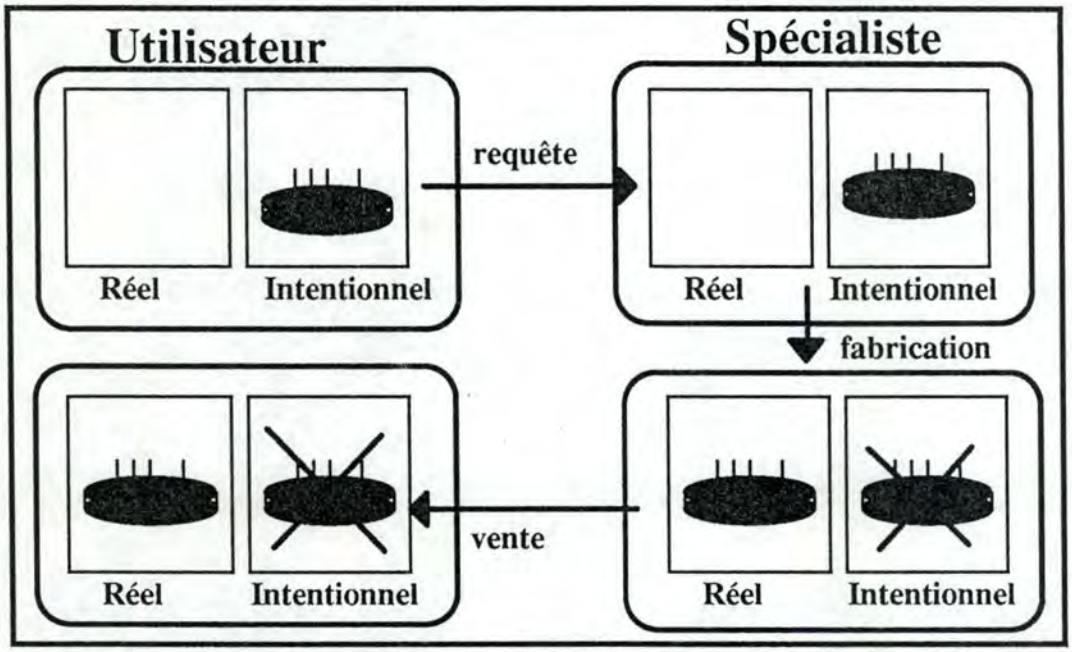


Figure 2.1 : Intentions et langage.

La figure 2.1 distingue deux parties, d'une part celle qui a des intentions sur une tâche mais qui ne sait pas directement agir dessus (l'utilisateur) et d'autre part, celle qui sait agir sur la tâche et qui va donc faire sienne les intentions de la première, pour la satisfaire (acteur, spécialiste).

Entre ces deux parties, on trouve une procédure complexe de traduction d'intentions en langage, de langage en actes et d'actes en intentions, le tout dans un but d'autorégulation du système.

Par la suite (chapitre 3), nous verrons en quoi il est intéressant d'introduire un troisième larron entre les deux premiers : l'interface.

2.3. La division du travail en LN

Avant de s'attarder plus avant sur les actes de langage, il est peut être bon de donner une illustration d'architecture modulaire de traitement du LN, et de préciser où nous nous situons dans le processus de traitement du LN.

2.3.1. Division en modules et communication entre modules : une architecture traditionnelle

Le traitement automatisé du LN est un processus complexe faisant intervenir plusieurs disciplines (linguistique, informatique, psychologie, ...). Face à ce type de problème, l'homme s'est convaincu au fil des travaux dans ce domaine de la nécessité de diviser le travail en

"spécialités" cohérentes tout en ne négligeant pas les possibles interactions entre ces "spécialités". Avec [Pierrel 87], [Haton 91], [Haton 92], [Sabah 89] (et bien d'autres), c'est devenu une réalité pour le traitement automatique de la parole.

Sans vouloir s'encombrer de définitions inutiles, précisons simplement, que de manière traditionnelle (pour ne pas dire traditionaliste), le traitement du LN est divisé de façon hiérarchique en spécialités : la phonétique, le lexique, la syntaxe, la sémantique et la pragmatique; chacune effectuant un certain type de traitement particulier, et nécessitant donc des types de connaissances adaptées, sur le signal en s'inspirant des résultats des traitements des modules inférieurs.

Au bas de l'échelle, on retrouve la phonétique dont le but est de "*passer d'une représentation dont les frontières sont imprécises, à cause des phénomènes de transitions entre sons et de co-articulations, à une représentation incertaine (incertitude quant à la présence de tel phonème sur telle zone du signal) plus utilisable pour les niveaux supérieurs*" ([Romary 89]).

Suit le lexique dont le but est de "*faire correspondre à des phonèmes des mots de la langue*", le tout également selon une certaine probabilité. Le module syntaxique vérifie la correcte construction des ces mots en phrases et de ce fait élimine déjà un certain nombre d'hypothèses.

La sémantique s'attache au sens des énoncés, et valide d'une certaine manière la syntaxe et tout ce qu'il y a en amont; la pragmatique quant à elle, vise à faire le lien entre un énoncé et l'univers du discours précédemment défini; c'est la science du contexte.

La démarche est alors la suivante à chaque niveau : on se base sur les conclusions du niveau inférieur, on effectue un certain nombre de traitements internes validant ou non les hypothèses inférieures, et on ressort avec un certain nombre d'hypothèses de niveau supérieur.

Dans la pratique, cette division du traitement du LN, a servi de base à la définition d'un certain nombre d'architectures (Myrtille I et II, Partner, ...).

A titre d'illustration, nous reprendrons l'architecture du système de reconnaissance du CRIN [Deville 89] :

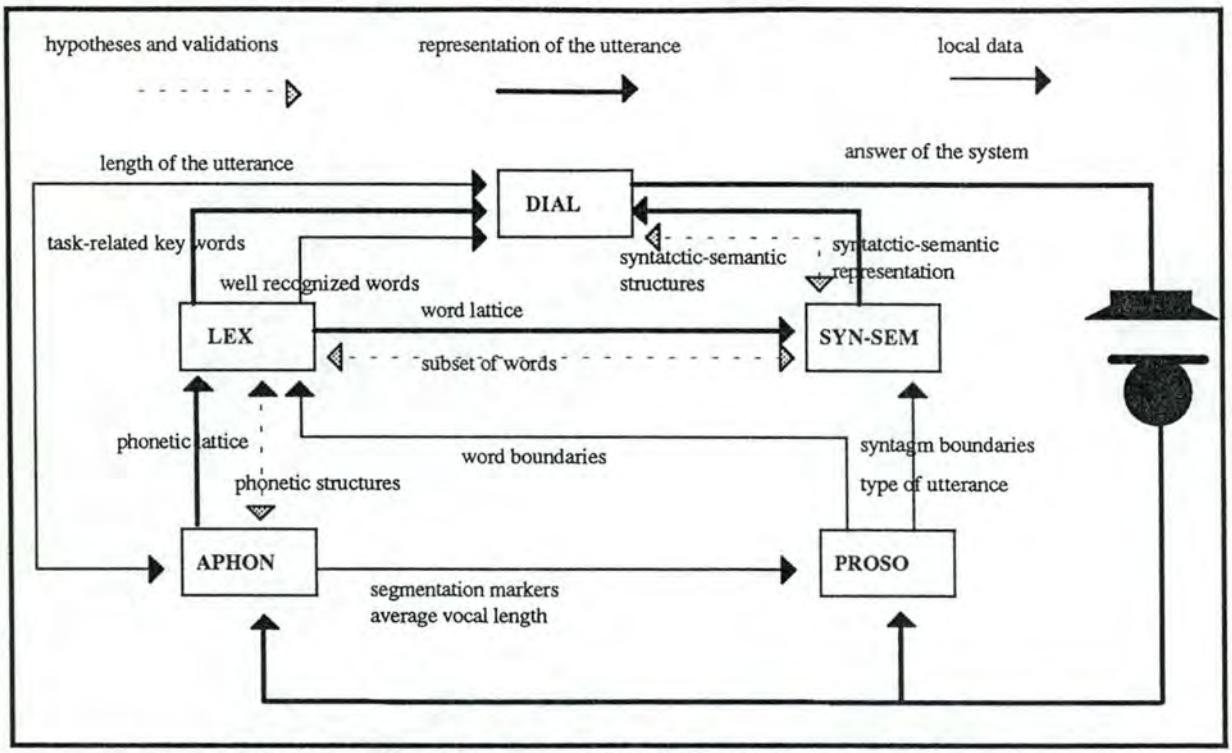


Figure 2.2 : Overall architecture of the CRIN dialogue system.

Si cette découpe en modules semble devoir être dépassée, la mise en évidence de différents niveaux dans le traitement du LN permet de mieux situer nos objectifs, et d'expliquer quels sont nos problèmes et quels sont ceux qui, pour nous, n'en sont pas ou plus.

2.3.2. Le niveau Pragmatique

En ce qui nous concerne, notre attention se portera tout particulièrement sur le niveau pragmatique.

Définir la pragmatique n'est pas chose aisée. Pour s'en convaincre, il suffit de se référer à [Levinson 83] et de remarquer qu'il met plus de 25 pages pour essayer de faire le tour d'une définition qui le satisfasse.

Si tout le monde est plus ou moins d'accord pour reconnaître que la pragmatique est la science du contexte, définir son champ d'application nous entraînerait hors sujet; nous nous contenterons donc de la définition (qui, il est vrai nous arrange bien) donnée par [Romary 89] :

La **pragmatique** est "*la science du contexte, c'est-à-dire celle qui précise comment une phrase peut se transformer en un* ***acte réel d'énonciation***".

La théorie des actes de langage permet d'éclairer les mécanismes de transformation d'une phrase, un énoncé, donc de l'information en un acte réel d'énonciation, et permet donc de faire le lien entre la traduction de langage en intentions et la logique d'actions qui est la nôtre.

2.4. Les actes de langage

Précisons d'abord le concept d'**acte réel d'énonciation**.

2.4.1. Les énoncés performatifs

Examinons les travaux d'Austin et de Searle sur les actes de langage. Bien que de nombreux ouvrages de synthèse reprennent ce concept ([Sabah 89], [Beardon 91]), nous nous inspirerons plus précisément de la synthèse des travaux d'Austin (et accessoirement de Searle) présentée au chapitre 5 de [Levinson 83].

Avant les travaux d'Austin sur le sujet[15], on considérait qu'un énoncé avait une signification s'il pouvait être vérifié (prendre la valeur vrai ou faux), cette démarche s'inspirant du positivisme (= **constative statements**).

Or, remarque Austin, il existe un certain nombre d'énoncés "*that they are not used just to say things, i.e. describe state of affairs, but rather actively to do things*".

Ainsi, "*je te baptise Hervé*", est plus qu'un simple énoncé de constation (pouvant être confirmé ou infirmé) : c'est un acte en lui même qui tout au plus peut être mauvais, inopportun (si la personne qui déclare cette phrase n'est pas prêtre ou que le bébé ne doit pas s'appeler Hervé, mais Jean, ...).

Ainsi, il nomme ce type d'énoncés, "**performative statements**" : "*unlike constatives, performatives cannot be true or false (given their special nature, the question of truth or falsity simply does not arise), yet they can go wrong*".

Se basant sur le fait que pour que ces énoncés deviennent des actes, il faut un certain nombre de pré-conditions, il définit des pré-conditions génériques qu'il appelle "*Felicity conditions*". Un **énoncé performatif** devient un acte valide si ces conditions sont respectées.

1. *The speaker said he would perform a future action.*
2. *He intends to do it.*
3. *He believes he can do it.*
4. *He thinks he wouldn't do it anyway, in the normal course of actions.*
5. *He thinks the adressee wants him to do it (rather than not to do it).*
6. *He intends to place himself under an obligation to do it by uttering U.*
7. *Both speaker and adressee comprehend U.*
8. *They are both conscious, normal human beings.*
9. *They are both in normal circumstances- not e.g. acting in a play.*
10. *The utterance U contains some IFID wich is only properly uteered if all the appropriate conditions obtain.*

---

[15] Austin, J.L. (1962). *How to do things with words.* Oxford: Clarendon Press.

Si on respecte le **principe de coopération** de Grice, ces pré-conditions sont quasi-automatiques.

Il poursuit, alors, son travail dans deux voies fortement liées.

Premièrement, il remarque qu'à côté des performatives traditionnelles ("*explicit performatives*") qu'il considère comme des sortes de cérémonies, il y a un nombre non déterminé de structures de phrases qui peuvent jouer le même rôle (ainsi pour "*je te baptise*": "*tu porteras le prénom de ...*", "*Hervé, tel sera ton prénom*", ...). Il appelle ces énoncés : Implicit performatives; ces énoncés pouvant toujours être ramenés à un performatif explicite.

Deuxièmement, partant du fait que la dichotomie constatif/performatif ne tient plus : en général, les énoncés comportent un peu des deux (" *' I warn you the bull will charge ' seems simultaneously to perform the action of warning, and to issue a prediction which can be assessed as true or false*"), il englobe sa théorie dans une théorie plus générale des actes de langage. "*So it is now claimed that all utterances, in addition to mean whatever they mean, perform specific actions (or do thing) ...*"

Il détermine trois types d'actions que portent tout énoncé :

- ***Locutionary act*** ou acte locutoire : l'action d'énoncer une phrase d'un certain sens. C'est l'acte de surface.

- ***Illocutionary act*** ou acte illocutoire ou encore Speech act : "*what is directly achieved by the **conventional force** associated with the issuance of a certain kind of utterance in accord with a conventional procedure*". L'élément principal de la définition est l'universalité de l'acte par rapport à l'énoncé : face à un acte illocutoire, tout le monde agit de la même manière, indépendamment du contexte d'énonciation.

- ***Perlocutionary act*** ou acte perlocutoire : "*is specific to the circunstances of issuance, and is therefore not conventionally achieved just by uttering that particular utterance, and includes all those effects, intended or unintended, often indeterminate, that some particular utterance in a particular situation may cause*".

[Maybury 92] permet de mieux situer la différence entre acte locutoire et illocutoire. Il s'agit essentiellement d'une différence de niveau d'abstraction. Pour illustrer ses propos, il présente une table reprenant quelques "surface speech acts" (figure 2.3).

Surface speech act	Syntactic form	Example
assert	declarative	"John and Mary walk to the park."
ask	interrogative (with question mark)	"Do John and Mary walk to the park ?"
command	imperative	"(John and Mary) walk to the park."
exclaim	imperative (with exclamation point)	"(John and Mary) walk to the park !"
recommend	declarative with auxiliary "should"	"John and Mary should walk to the park."

Figure 2.3 : Surface speech acts.

Cette table illustre comment une proposition que nous noterons : (walk (john, mary, park)) peut être réalisée par différents actes de surface en fonction de l'objectif de la conversation. Par exemple, *assert* peut être réalisé par une déclarative; *ask*, par une interrogative, ...

"In contrast to locutionary acts, illocutionary acts are more abstract representation of the linguistic action performed by an utterance and may be associated with more than one type of locutionary act."([Maybury 92, pp. 6])

Les actes de surface : *command, exclaim* et (dans une moindre mesure) *recommend* de la figure 2.3, se rattachent à la même fonction : on demande à John et à Mary d'aller marcher dans le parc. Ce type de fonction a été regroupé par Austin et Searle sous le nom de requête et constitue un acte illocutoire.

Les actes locutoires, en plus de référencer les actes illocutoires, véhiculent l'information sur la manière dont ils effectuent ces références (demande plus ou moins polie, ...). Ce type d'information passe au second plan lorsqu'on développe des système de reconnaissance du LN par une machine; nous centrerons donc notre étude sur les actes illocutoires.

Le travail qui suivit ces définitions fut de recenser les actes illocutoires génériques, et de définir les pré-conditions à leur correcte exécution.

Face aux actes illocutoires génériques, on retrouve un certain nombre de constructions d'énoncés qui ont la même force illocutoire. Searle appelle ces énoncés des *"indirect speech acts"*. Le problème central est alors de retrouver la force illocutoire de chaque énoncéet donc de le rattacher à un acte générique.

Nous ne reprendrons pas la liste des actes illocutoires, perlocutoires, ... dans ce travail; ce serait inutile et fastidieux. Le point suivant justifie ce choix.

2.4.2. Quels actes de langage pour un langage de commande?

Qu'allons nous retenir de tout cela ?

Tout d'abord, en ce qui concerne l'acte locutoire, nous n'en dirons rien, ou très peu : c'est l'acte de surface d'énonciation. Il nous permettra tout au plus de faire le lien entre les énoncés (actes locutoires de surface) et les actes illocutoires.

Comme déjà annoncé, à un même acte illocutoire correspondent plusieurs actes de surface, ce qui pose le problème du choix de l'acte illocutoire. Nous l'éviterons en considérant que nous travaillons au niveau illocutoire (et au delà).

Ensuite, autant l'acte illocutoire doit être universel pour être ce qu'il est, autant l'acte perlocutoire dépend des circonstances de l'énonciation. Ainsi, la phrase *"je t'interdis de faire çà "*, déclenchera un acte illocutoire d'interdiction, mais suivant la façon dont elle est dite, la situation des acteurs, ... elle pourra déclencher un acte perlocutoire d'obligation d'agir, de peur, ...

Pour un langage de commandes, l'acte perlocutoire doit être éludé : la machine doit se contenter exclusivement de l'acte illocutoire contenu dans l'énoncé; il serait en fait regrettable

que, suivant son humeur (ou que sais-je?), la machine produise un autre résultat que celui qui est (expressément) demandé.

Enfin, l'acte illocutoire étant un acte, dans une optique de validation d'exécution (et de planification), il convient de définir un certain nombre de pré-conditions à son exécution.

L'acte de langage central d'un langage de commandes est la commande (on s'en serait douté!) ou requête. Nous tenterons donc d'en donner une définition opérationnelle et de faire le lien entre locutoire, illocutoire et perlocutoire. Nous nous intéresserons également à l'acte illocutoire d'information (Inform) et nous montrerons comment on peut modéliser une question par une requête d'information.

2.4.3. Définition opérationnelle de certains actes de langage

Typiquement, dans un dialogue de commande, l'utilisateur émet une requête initiale (REQUEST), la machine interprète celle-ci et l'exécute ou pose une question si elle n'en a pas saisi toute l'étendue; en réponse à cette dernière, l'utilisateur produit de l'information (INFORM).

Les opérateurs de croyance

Les actes de langage sont des actes qui agissent non pas directement sur l'environnement, la réalité, mais plutôt sur le monde de la subjectivité, les mondes intentionnels[16]. On définit généralement un certain nombre d'opérateurs de croyances suffisants[17] pour représenter le monde des intentions de chaque individu. Nous reprendrons la notation utilisée par [Allen 80], [Cohen 79], [Litman 87], et bien d'autres. Le monde intentionnel est défini grâce à trois types de prédicats : Know, Believe et Want.

On se contentera des pseudo définitions suivantes : (pour des définitions plus convaincantes, on peut se référer à [Sadek 87] par exemple).

Want (acteur, action) ≡ l'acteur veut effectuer l'action.

Believe(acteur, fait) ≡ l'acteur croit le fait.

Know(acteur, fait) ≡ Believe(acteur, fait) & le fait est vrai.

On ajoute en plus les prédicats suivants :

Know-if (acteur, fait) ≡ Believe(acteur, fait) si le fait est vrai

Believe(acteur, Not fait) si le fait est faux

---

[16] Ce n'est pas toujours le cas : "Je te rejette comme épouse", répété à trois reprises, aura certainement un effet sur la subjectivité des mariés, des parents, ... mais, s'il est fait devant témoins dans certaines cultures musulmanes, aura surtout l'effet réel d'entériner le divorce du couple concerné.

[17] On s'en contentera.

Know-ref (acteur, terme, proposition) ≡ l'acteur connaît une description d'un terme qui satisfait la proposition (la rend vraie) [Litman 87].

Cando (acteur, action) ≡ l'acteur est physiquement en mesure d'effectuer l'action.

Vers une définition opératoire

Définissons les actes de langage rencontrés dans le dialogue Homme-Machine en essayant d'illustrer les aspects locutoires, illocutoires et perlocutoires introduits par Austin et Searle.

Nous utiliserons le formalisme suivant :

Nom-acte (arg $_1$**, ...,** **arg** $_n$**)** ≡ L'acte **Nom-acte** a les arguments **arg** suivant.

Pré-condition:

- **cando pr** ≡ La cando pré-condition reprend l'ensemble des conditions sur l'état mental du locuteur nécessaires au bon accomplissement de l'opération.

- **want pr** ≡ La want pré-condition met l'accent sur le caractère intentionnel de ce type d'opération.

Post-condition ≡ Regroupe l'ensemble des modifications qu'entraîne l'exécution de l'acte sur les mondes réels et intentionnels.

Contrainte ≡ Conditions qui doivent être respectées pour permettre l'exécution de l'acte, indépendemment de toute autre action préalable (par ex : le récepteur doit écouter l'émetteur). On utilisera peu cette section, nous croyons en effet que la condition de coopération entre acteurs valide automatiquement ces dernières.

Décomposition ≡ Reprend l'ensemble des "actions de niveau inférieur" dans une séquence donnée (un plan) qui découlent de l'action spécifiée

Nous exprimerons les conditions et contraintes sous forme de prédicats, en utilisant Believe, Want, Cando, Know, Know-if, Know-ref précédemment définis.

Les commentaires seront précédés d'un ; .

La Requête

Request (speaker, hearer, action)

; le speaker demande à l'hearer de déclencher l'action.

Pré-condition:

cando: Believe(speaker, Cando (hearer, action))

& Believe(speaker, Believe(hearer, Cando(hearer, action)))

want: Believe(speaker, Want(speaker, request-instance))

Post-condition: Believe(hearer, Believe(speaker, Want(hearer, action)))

Contrainte: Agent(action, hearer)

Décomposition:

; il s'agit ici de faire le lien avec les actes de langage de surface (actes locutoires).

;Une solution serait de reprendre la décomposition proposée par [Litman 87] :

1) **Surface-request**(speaker, hearer, action)

2) **Surface-request**(speaker, hearer, Informif(hearer, speaker, Cando(hearer, action)))

3) **Surface-inform**(speaker,hearer, not(Cando(speaker, action)))

4) **Surface-inform**(speaker, hearer, Want(speaker, action))

et on pourrait facilement en trouver d'autres...; et définir ensuite (en donnant une énumération la plus exhaustive possible des structures de phrases admises) ces actes de surface.

De notre côté, nous supposerons qu'il existe un ou plusieurs modules d'analyse syntaxico-sémantique (et pragmatique) de l'énoncé qui activent l'acte (ou les actes) de langage qui correspondent; nous ne développerons donc pas la partie "décomposition" de la définition, ces modules étant pour nous des boîtes noires.

Alors que la partie décomposition de la définition est sensée faire le lien avec l'acte locutoire (ou encore acte de surface) défini par Austin (*locutionary act : the utterance of a sentence with determinate sense and reference*), l'aspect perlocutoire de l'acte apparaît lorsqu'on définit l'opération logique **Cause-to-Want**. En fait, nous réagissons tous plus ou moins différemment à un acte de langage (acte illocutoire). En définissant une fois pour toutes l'acte perlocutoire perçu de l'interface, on évite cette indétermination.

Cause-to-Want(speaker, hearer, act)

Pré-condition:

cando: Believe(hearer, Believe(speaker, Want(hearer, act)))

Post-condition: Believe(hearer, Want(hearer, act))

Austin donne la définition suivante de l'acte perlocutoire : The bringing about of effects on the audience by means of uttering the sentence, such effects being special to the circumstances of utterance [Levinson 83].

Nous supposerons que la machine est dépourvue de sentiments humains comme la peur, la colère, la pitié, ... Dès lors, lorsqu'elle sait qu'une personne veut qu'elle fasse quelque chose, en automate serviable qu'elle est, elle va vouloir satisfaire son locuteur en voulant faire cette chose, et ne s'embarrassera pas de considérations affectives.

Comme le note Cohen et Perrault : "to get someone to do something, one need only get that person to know that you want then to do it."[Cohen 79].

En appliquant cette nouvelle opération juste après le **Request**, la post-condition du Request devient :

```
Post-condition: Believe(hearer, Want(hearer, action)).
```

La production d'information

La production d'information, n'est a priori pas un acte de langage; pour Austin, il s'agit simplement d'un énoncé constatif. Néanmoins, dans une théorie plus générale des actes de langage (Searle) et en supposant que pour modifier un état en général, donc un état mental en particulier, il faut produire un acte, on considère que ce type d'énoncé a également une valeur performative.

```
Inform(speaker, hearer, prop)
; prop est une information qui a la valeur logique vraie.[18]
Pré-condition:
 cando: Believe(speaker, prop)
 want: Believe(speaker, Want(speaker, inform-instance))
Post-condition: Believe(hearer, Believe(speaker, prop))
```

De la même manière, on fait le lien avec l'acte perlocutoire en définissant une opération qui suit le **Infom** :

```
Convince(speaker, hearer, prop)
Pré-condition:
 cando: Believe(hearer, Believe(speaker, prop))
Post-condition: Believe(hearer, prop).
```

[18] Par exemple prop peut être "le chien est mort", que l'on peut traduire sous la logique du premier ordre par : Mort (chien).

Nous ne retenons comme seul but sous-entendu par un **Inform** d'un locuteur vers un récepteur, que ce récepteur croit ce qui est dit (prop) et pas, par exemple, lui fait peur ou envie ou ...

En appliquant le **Convince** juste après le **Inform**, on peut réécrire la post-condition de ce dernier comme suit :

Post-condition: Believe(hearer, prop)

Avec le **Inform** défini ci-dessus, on ne peut exprimer que des vérités. Comme nous le verrons, une **question** est en fait un **Request** dont l'action est un **Inform**. On doit donc en plus de l'**Inform** classique, définir un **Informref** et un **Informif** pour pouvoir exprimer les questions sous forme de requête.

Informref(speaker, hearer, term, prop)

; le speaker dit au hearer que le terme term satisfait

; la proposition prop. Il dit toujours quelque chose de vrai mais met l'accent sur le term qui fait que cette proposition est vraie.

Pré-condition:

cando: il existe term tq Knowref(speaker, term, prop)

want: Believe(speaker, Want(speaker, Informref))

Post-condition: Il existe term tq Believe(hearer, Knowref(speaker, term, prop)

Contrainte: Parameter(term, prop)

On met en évidence l'acte perlocutoire en définissant :

Convince-ref(speaker, hearer, term, prop)

Pré-condition:

cando: Il existe term tq Believe(hearer, Know-ref(speaker, term, prop)).

Post-condition: il existe term tq Knowref(hearer, term, prop)

Contrainte: Parameter(term, prop).

En appliquant **Convince-ref** à la suite de **Informref**, on redéfinit la

Post-condition: il existe term tq Knowref(hearer, term, prop).

```
Informif(speaker, hearer, prop)
Pré-condition:
 cando: Knowif(speaker, prop)
 want: Believe(speaker, Want(speaker, informif))
Post-condition: Believe(hearer, Knowif(speaker, prop))

Convince-if(speaker, hearer, prop)
Pré-condition: Believe(hearer, Knowif(speaker, prop))
Post-condition:  Knowif(hearer, prop)
```

On retrouve comme post-condition pour **Informif**:

```
Post-condition: Knowif(hearer, prop)
```

La question

La **question** est donc modélisée par :

```
Request (speaker, hearer, Inform (hearer, speaker, prop)).
```

Bien que cette définition semble légitime, elle ne sera jamais appliquée comme telle pour la bonne et simple raison que pour émettre un tel acte, le speaker doit déjà connaître la valeur de **prop**.[19]

Si on connaît la valeur de **prop**, cela ne sert a rien de poser la question. A partir de là , on peut définir deux types de questions : les *oui/non questions* et les *Wh-questions*.

Les *oui/non-questions* sont des questions dont les réponses possibles se limitent à "oui" ou "non". Elle sont modélisées par :

```
Request (speaker, hearer, Infom-if (hearer, speaker, prop))
```[20].

Les *Wh-question* (Wh comme what, who, which, ...) sont des questions qui demandent de préciser la valeur d'un terme pour qu'une proposition soit vraie. On les modélise par :

---

[19] Typiquement, cela modélise des phrases du genre : " dis-moi que le chien est mort ?".

[20] On modélise de la sorte les énoncés du genre : " dis-moi si le chien est mort ?", "le chien est-il mort?", ..

```
Request (speaker, hearer, Informref (hearer, speaker, term,
prop)).
```

On n'en dira pas plus, le but de ces définition et de l'introduction des actes de langage dans notre travail se limitant à faire le lien entre monde intentionnel, langage, acte et application. Les définitions formelles présentées plus haut seront utilisées implicitement à un méta-niveau lors de la reconnaissance des énoncés de l'utilisateur (un request aura comme effet de placer une zone hypothétique de contexte infini dans l'historique du dialogue (Cfr. Chapitre 5). On s'intéressera davantage à ces constructions, lorsqu'on parlera de la génération de réponses via différents modes (Cfr. Rhetorical acts au Chapitre 6).

2.5. Conclusion

Le monde réel, composé principalement du modèle de la tâche reprend les connaissances sur le monde et sur la situation (Figure 1.1). Pour faire évoluer la réalité, donc le monde réel, il faut agir; ces actions devant rendre la réalité conforme aux objectifs, intentions reprises dans le monde intentionnel de l'acteur.

Dans le type de problème qui nous occupe (pilotage d'une application informatique par une interface comprenant le LN), l'utilisateur du système n'est pas l'acteur, il ne sait pas agir directement sur la tâche. Il va donc devoir traduire ses intentions en langage : émettre un énoncé langagier comprenant au minimum un acte de langage de requête sur ces intentions.

L'exécution de cet acte aura comme effet, chez l'interlocuteur (qui lui, sait agir sur la tâche), de faire en sorte que les intentions de l'émetteur devienne siennes. Il ne lui restera alors plus qu'à agir sur la réalité et modifier en conséquence le monde réel perçu par l'utilisateur pour réaliser leurs intentions communes sur la tâche.

La Langage Naturel constitue donc bien un moyen d'action (différé ou à distance).

Chapitre 3

3. Interaction et dialogue

Concepts : **interface, modèle de l'application, dialogue, plan de la tâche.**

Si le langage de commande est un moyen d'action (*Remote Action Medium*), la notion de dialogue de commande ne prend tout son sens que lorsqu'il y a interaction.

Pour permettre cette interaction entre l'homme et la machine, il est nécessaire d'introduire une entité dans l'odinateur entre utilisateur et application : l'interface.

Ce chapitre sera consacré à cibler cette interface. Quel est son rôle ? Comment peut-elle influencer la façon dont l'utilisateur communique ? Quelles sont ses relations avec l'utilisateur et la tâche ? Pourquoi est-il nécessaire d'y intégrer une modélisation de la tâche et un module de planification pour permettre une interaction en LN ? Qu'en est-il de Multiworks ?

3.1. Pourquoi une interface ?

Dans la vie de tous les jours, il est rare que l'on s'adresse directement aux producteurs de services[21].

L'outil informatique a la particularité de rassembler plusieurs centres de traitements, aux fonctionnalités diverses, sous le même capot de machine. S'il on veut éclairer le fonctionnement des ordinateurs, il est indispensable d'arrêter de considérer la machine comme **une seule entité.**

Jusqu'à présent, nous avons réduit notre univers aux seuls entités utilisateur et machine. Il est plus que temps de séparer la composante tâche de celle qui la commande : l'interface; et cela pour plusieurs raisons.

---

[21] En guise d'illustration, reprenons l'exemple du chapitre précédent (figure 2.1). Lorsqu'on rentre dans une pâtisserie, jamais personne ne commande de gâteaux au pâtissier directement. On passe toujours par un intermédiaire dont le but est de servir d'interface entre les clients et le technicien (une vendeuse par exemple).

Premièrement, il est beaucoup plus facile pour tout développeur de système de séparer les aspects techniques de la tâche, de la façon dont on va la piloter, les compétences demandées à chacune de ces spécialités étant fort différentes (technicien versus ergonome).

Deuxièmement, en retirant les aspects communications avec les utilisateurs des produits, on simplifie la production des outils, on augmente la portabilité; bref, on accroît son marché potentiel. L'éditeur MULTICARD que doit piloter Multiworks a par ailleurs été développé par des équipes tout à fait étrangères au CRIN.

Enfin, mais on pourrait encore trouver d'autres arguments, cette séparation permet de faire évoluer séparément l'interface et l'outil qu'elle commande.

Une **interface** (Homme-Machine) est une entité qui permet les échanges entre un utilisateur et une application. Dès lors, toute communication entre ces deux entités doit passer par elle.

3.2. Modèle d'interaction

Nous reprendrons, dans cette section, un modèle d'interaction entre un homme et une machine qui permet de cibler les problèmes propres à la réalisation d'une interface.

Nous nous attacherons à présenter une instanciation de ce modèle propre à l'interaction en LN finalisé.

3.2.1. Vue classique

La figure 3.1 reprend "le modèle de ~~la tâche~~ (l'interaction) de D.A. Norman", [Bodart 90].

Remarque : Norman utilise des termes pour lesquels nous avons une acception différente. Nous prenons donc la liberté d'en utiliser d'autres plus proches de notre réalité et d'apporter quelques modifications au modèle...

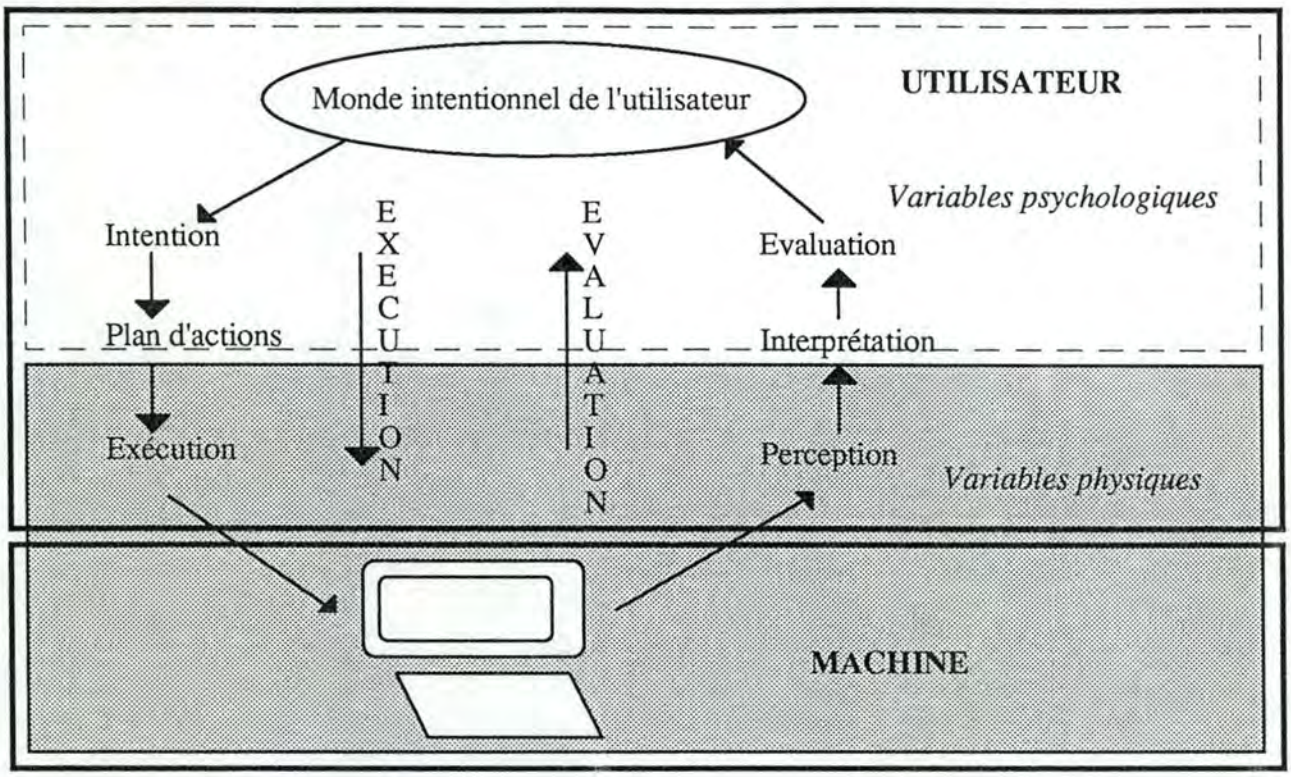


Figure 3.1 : Modèle de l'interaction de Norman.

Pour Norman, la difficulté principale de l'interaction consiste à "*réduire le fossé qui sépare l'univers psychologique dans lequel une personne pense la tâche à accomplir ... et l'univers physique de l'interface dans lequel elle accomplit les actions nécessaires à la réalisation de la tâche*" [Bodart 90].

L'interaction entre un utilisateur et une machine peut être modélisée par une boucle itérative composée de 7 étapes :

• **Etablir l'objectif de l'interaction** : l'utilisateur a des intentions sur la tâche à accomplir. Il les regroupe dans ce qu'on a appelé son monde intentionnel.

• **Déterminer un sous-objectif, une intention particulière** : l'utilisateur décompose son objectif en sous-objectifs cohérents.

• **Spécification du plan d'actions** : "*Afin de spécifier la séquence d'actions, l'utilisateur doit :*

- *traduire ses objectifs et intentions en états désirés du système (.....);*
- *déterminer les dispositifs des mécanismes de contrôle qui produiront cet état (...);*
- *déterminer les manipulations requises de ces mécanismes.*

Le résultat consiste en une spécification mentale des actions qui doivent être exécutées." [Bodart 90].

Cette étape de l'interaction oblige l'utilisateur à savoir comment l'interface représente ses connaissances (sémantique des objets de l'interface) et à connaître les possibilités d'action de celle-ci.

• **Exécution de la séquence d'action** : l'utilisateur interagit avec l'interface.

• **Perception des résultats** : l'utilisateur, pour évaluer son interaction, a besoin de voir comment la tâche a évolué. L'interface lui présente les résultats dans son mode de représentation des choses.

• **Interprétation** : l'utilisateur traduit dans son mode de pensée les informations données par l'interface.

• **Evaluation** : en fonction des résultats, l'utilisateur modifie son objectif initial.

Par variables psychologiques, on entend la façon dont l'utilisateur perçoit la réalité.

Par variables physiques, on entend la façon dont l'interface perçoit la réalité.

Dans ce schéma, on fait abstraction de la tâche : on considère que l'ordinateur est avant tout composé d'une interface avec laquelle on va interagir. La tâche ne sert que de domaine aux objectifs; en aucun cas, on a voulu modéliser son fonctionnement.

3.2.2. Interaction en LN

Lorsque le style d'interaction entre l'utilisateur et la machine est le LN finalisé, le schéma de la figure 3.1 prend une autre dimension (Figure 3.2).

Le LN finalisé, malgré les difficultés de traitement sous-jacentes à ce mode de communication (Cfr. Chapitre 1), est un bon moyen d'interaction entre un homme et une machine et permet de définir un espace de communication cohérent.

Nouveau modèle

L'homme s'adresse en LN à une machine (et inversement) pour exprimer ses intentions, idées de la façon la plus naturelle pour lui, la plus proche de son mode de pensée (variables psychologiques).

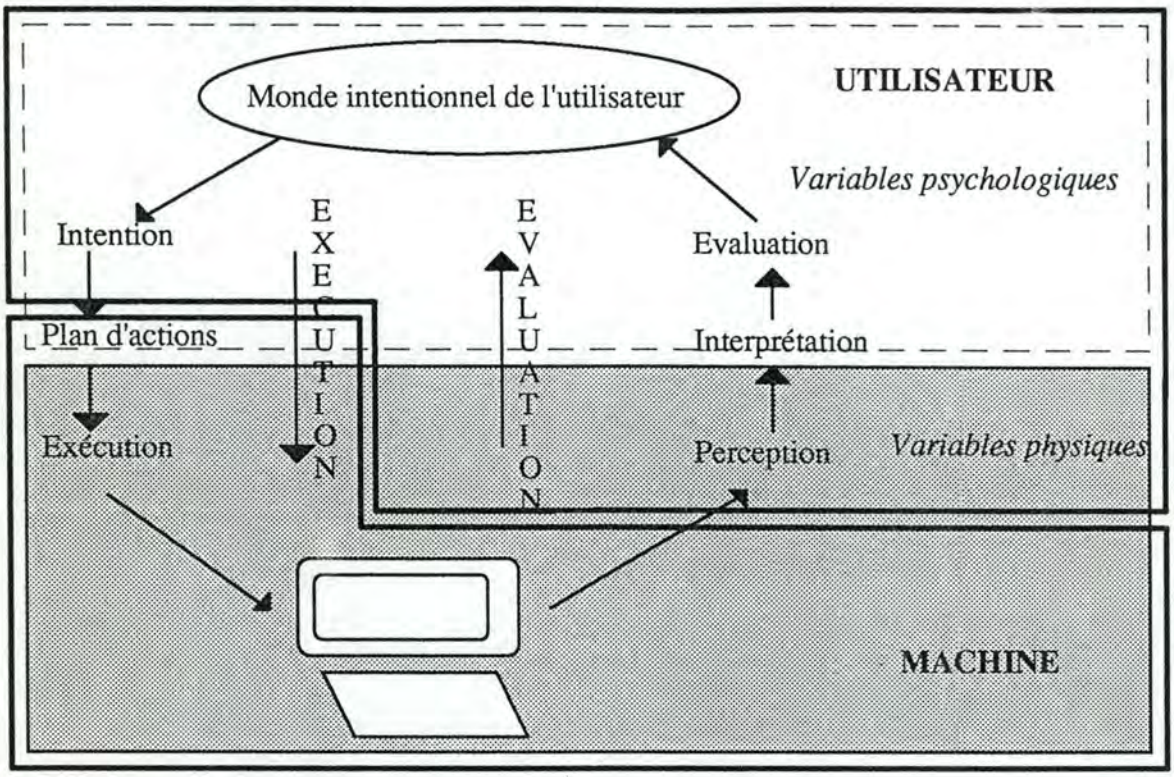


Figure 3.2 : Interaction en LN finalisé.

En utilisant le LN, on veut que l'homme ne se soucie plus de la façon dont la machine perçoit la réalité (Cfr. Figure 1.2); on veut réduire le plus possible la distance entre variables physiques et variables psychologiques, pour qu'à la limite, les variables physiques disparaissent chez l'utilisateur. On doit donc pouvoir se passer de la phase d'exécution : la simple énonciation des spécifications d'actions doit suffire pour agir sur la tâche (Cfr. actes de langages).

Pour permettre ce type d'interaction, il faut que l'interface dispose d'une modélisation de l'univers analogue à celle de l'homme (variables psychologiques). Dès lors, on peut vouloir reporter la spécification de la séquence d'actions sur l'interface et se contenter d'exprimer ses intentions en LN.

Les phases en elles-mêmes ne disparaissent pas, elles sont seulement déplacées de l'utilisateur à la machine. Le passage du psychologique au physique est alors à la charge de celle-ci

Dans l'autre sens, le projet qui nous occupe ne prévoit pas d'utiliser le LN pour présenter les résultats de la machine à l'utilisateur (présentation à l'écran, ...). On ne modifiera donc pas le modèle. La présence chez l'utilisateur des variables physiques et psychologiques reste donc indispensable.

Cohérence

Le Langage Naturel est-il un bon moyen d'interaction entre un utilisateur et une application ? Il est peut-être un peu tard pour se poser cette question, puisque la simple rédaction de ce travail suppose que oui. Néanmoins, les différents éléments de réponses que

nous avons isolés, laissent penser que, d'une part, l'utilisation du LN augmente, par sa nature, le naturel et la flexibilité dans le pilotage de l'application ("*natural langage emphasize naturalness and flexibility*" in [Capindale 90]) et, d'autre part, s'adapte particulièrement bien à l'ajout d'autres modes ou médium de communication ([Romary 91]).

Nous avons dit que toute communication en LN nécessitait une connaissance commune du code utilisé (Figure 1.1). La première partie du chapitre 1 a été consacrée à montrer que le LN était un moyen de communication complexe. Est-il pour autant compliqué voir ambigu car peu structuré a priori ? On peut penser que oui. Les recherches de ces vingts dernières années ainsi que les systèmes développés[22] tendent à montrer que, lorsque le LN est finalisé, c'est un moyen de communication plutôt clair et qui s'adapte particulièrement bien à la tâche qu'il est sensé "guider" ([Martin 89], [Hauptmann 88]).

Ce qui est problématique avec le LN, c'est la liberté dont jouit l'utilisateur. Il existe en effet presque une infinité de façon de demander la même chose. Les faibles taux de reconnaissance de la majorité des systèmes de traitement du LN sont le résultat de leur impossibilité à traiter correctement cette variété de constructions langagières.

Pour [Zoltan 91], il existe deux façons pour maximiser la probabilité qu'un système de traitement du LN reconnaisse les énoncés des utilisateurs :

- Programmer l'ordinateur à décoder toutes les façons dont les humains construisent leurs inputs.

- Limiter le nombre de constructions langagières de l'utilisateur.

La première méthode, quoique très tentante (traitement du LN indépendant de la tâche, donc universel) est actuellement et pour encore longtemps certainement, inaccessible.

La seconde approche consiste à laisser l'utilisateur s'ajuster aux limites du système. Elle peut être rencontrée de deux façons.

- Ouvertement : on indique à l'utilisateur l'ensemble des mots, des constructions possibles, ... avant utilisation du système.

- "Covertly" (de façon déguisée) : "*According to Becker (1975), people have a phrasal lexicon consisting of six major categories of lexical phrases. In creating their natural-langage communication, people reffer to this lexicon and "stitch together swatches of a text that they have heard before" (p. 38). The goal in person computer interaction is to capitalize on this stitching process so the combined swatches remain consistent ...*" ([Zoltan 91]). "*Diverses expériences, en particulier celles présentées dans GRECO (1985), confortent l'idée d'une adaptation du locuteur au niveau de langage utilisé par la machine ...*" ([Pierrel 89]).

Deux des principes fondamentaux que doit respecter toute interface homme-machine sont l'indépendance entre l'interface et l'application, et la cohérence dans le(s) moyen(s) de communication de l'interface [Scapin].

---

[22] Pour une une liste de ces applicatons, on peut par exemple se référer à [Pierrel 87], [Haton 91].

Ce principe de cohérence trouve tout particulièrement sa place dans les systèmes de reconnaissance du LN "*covertly restricted*". En limitant le nombre d'expressions, le vocabulaire ainsi que la façon de construire les réponses de la machine, on indique à l'utilisateur la façon optimale dont se déroule la conversation. Celui-ci, voulant, en principe, être coopératif, s'adaptera à ces restrictions implicites ([Chantraine 90], [Capindale 90]).

Dans le type d'application qui nous occupe, la majeur partie des réponses de la machine ne se feront pas par langage. En réponse à un ordre, on exécutera une série d'actions. Le langage utilisé par l'utilisateur s'aligne sur la nature des ces actions. Pour Multiworks, on peut donc s'attendre à des expressions assez directes et précises.

Lorsqu'on introduira d'autres modes d'interaction, il faudra veiller à les traiter sur un même pied (conserver la cohérence entre modes). La présentation des rhétorical acts au chapitre 6 va dans ce sens.

3.3. Dialogue interne et externe

La décomposition de la machine en deux entités autonomes et le passage obligé par l'interface de tout échange entre utilisateur et application permet de distinguer deux types de dialogue.

3.3.1. Vue classique

Par **dialogue interne**, on entend les échanges à l'intérieur de la machine entre interface et application.

Par **dialogue externe**, on entend les échanges entre utilisateur et interface (Figure 3.3).

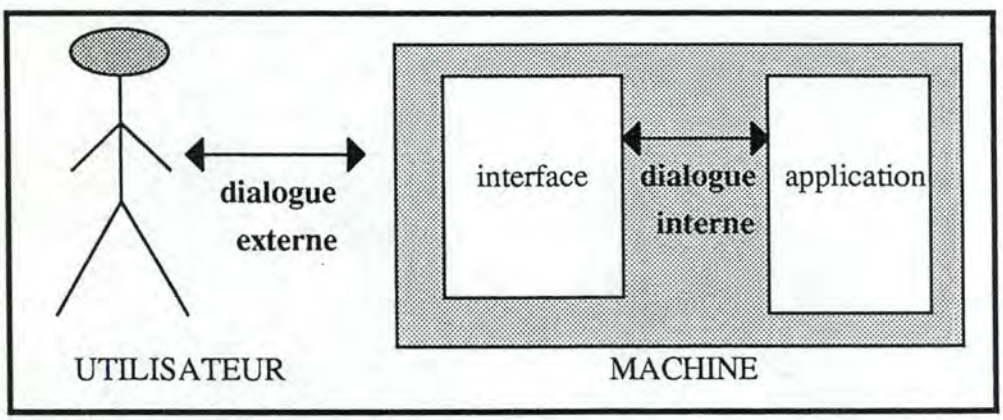


Figure 3.3 : Dialogue interne et externe.

Le dialogue interne ne nous intéressera que fort peu; notre attention se portera plus particulièrement sur le dialogue externe. Considérons dans un premier temps que le dialogue externe se fait uniquement en LN finalisé, présentons un modèle de gestion de ce dialogue, adaptons le au dialogue de commande d'application et justifions ainsi nos choix.

3.3.2. Une modélisation du dialogue HM en LN finalisé

Retenons les idées exprimées par [Luzzati 90] et tentons de les adapter au cas qui nous occupe. Le but de Luzzati est de modéliser les étapes d'un dialogue en LN en vue de le stopper ou de le re-diriger si on sent qu'il n'aboutira pas. Nous nous contenterons d'une vision des choses plus a posteriori, qui nous servira à arrêter nos choix.

Bien que le corpus traité soit un logiciel de renseignements horaires SNCF (qui n'a pas grand chose à voir avec Multiworks), nous pensons, que la classification des différentes interactions peut-être maintenue. Préférant le terme **requête** à celui de **question**, nous nous plierons à la terminologie utilisée par Luzzati. Une requête est donc vue comme une question, et le résultat de cette requête est une réponse.

Concepts de base

Reprenons la classification des Q/R (Questions/Réponses) proposée par Luzzati:

- Les questions/réponses principales (QP/RP) constituent le cadre de la description puisqu'elles définissent un espace de référence. On change d'espace de référence lorsqu'on change de QP.

- Les questions/réponses secondaires (QS/RS) se limitent à modifier un des éléments de la QP; et elles appellent toujours une RS de même nature que la RP.

- Les questions réponses incidentes (QI/RI) qui sont en l'occurrence des demandes de précision, de re-formulation ou de confirmation. Elles peuvent être le fait de la machine ou de l'utilisateur. Les QI/RI peuvent, enfin, être enchâssées ce qui permet de rendre compte de configurations relativement complexes.

Luzzati introduit alors deux axes sur lesquels se déplacent les Q/R principales et incidentes :

- L'axe régissant : reprend le parcours d'une QP à une RP.

- L'axe incident : correspond aux couples QI/RI qui peuvent aussi bien se suivre que s'enchâsser. Les QI/RI peuvent être l'objet de l'utilisateur ou de la machine indépendamment.

- Les QI opèrent un déplacement vers la gauche sur l'axe régissant et vers le bas sur l'axe incident.

- Les RI opèrent un déplacement vers la droite sur l'axe régissant et vers le bas sur l'axe incident.

- Les QP et RP opèrent un déplacement vers la droite sur l'axe régissant.

Exemple

Utilisateur 1 : je voudrais que tu déplaces la petite fenêtre. (QP)

Ordinateur 1 : de quelle petite fenêtre veux tu parler ?(QI)

U2 : de la petite rouge (RI)

O2 : celle avec le bord bleu ou vert ?(QI)

U3 : bleu (RI)

O3 : ou dois-je la mettre?(QI)

U4 : mettre quoi? (QI)

O4 : où dois-je mettre la fenêtre bleu ? (RI)

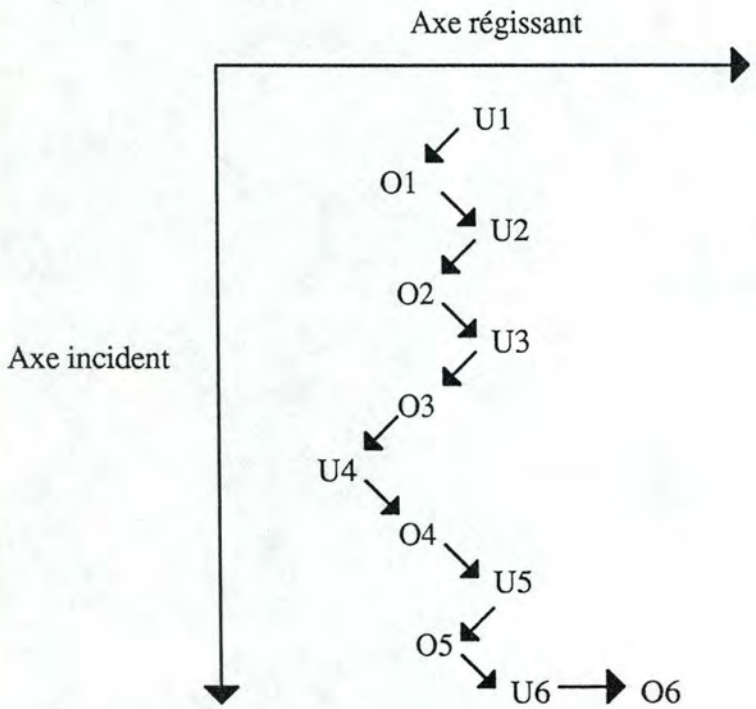
U5 : à gauche de la fenêtre verte. (RI)

O5 : ici, çà va ? (et il déplace la fenêtre à un endroit précis) (QI)

U6 : oui, merci. (RI)

O6 : (il laisse la fenêtre à cet endroit précis) (RP)

on obtient le tracé suivant :



La place des QS/RS

Le modèle prend tout son sens lorsqu'on tente d'intégrer les QS/RS dans le système d'axes. Deux situations opposées semblent possibles :

On considère les QS/RS comme des QP/RP

Chaque requête, question, est considérée comme un problème à part entière demandant une réponse complète et est donc formulée avec toute l'information indispensable à sa correcte

interprétation. Si cette réponse ne peut être obtenue, on préfère sauter la question plutôt que de tenter de l'éclairer par un dialogue incident.

On se situe alors dans une perspective de réponses maximalistes. On obtient un dialogue fort contraignant, rigide et somme toutes peu naturel (on oblige l'utilisateur et la machine à fournir des questions et réponses complètes à chaque tour de parole) mais qui est certain d'aboutir.

On considère les QS/RS comme des QI/RI

On se situe alors dans l'optique inverse; on gagne en naturel mais on risque de se perdre dans son discours.

Comme toujours, le juste choix se trouve au milieu. Il faut pouvoir laisser évoluer le dialogue tout en empêchant qu'il ne dérive. Luzzati propose de réaliser cet arbitrage en fixant le niveau d'incidence possible.

Nous pensons que la souplesse du système doit aussi dépendre de l'instigateur de l'incidence.

Dans le cas où c'est la machine qui pose une question ou qui fournit une réponse, on doit s'attendre à ce que cette réponse soit aussi précise que ne le souhaite l'utilisateur; ce qui, en principe doit annuler l'incidence (si l'interprétation des intentions de l'utilisateur est correcte, il doit être content directement des réponses fournies). Nous nous dirigerons donc vers un maximalisme des réponses (Dans le cas où la réponse est le déplacement d'une fenêtre, on ne voit pas bien à quoi correspondrait une réponse minimale...) et des questions.

Si l'ordinateur doit être irréprochable, il n'en est pas de même de l'utilisateur. N'oublions tout de même pas que c'est justement parce que l'utilisateur n'exprime pas explicitement ses intentions dans le dialogue que les systèmes de gestion du LN sont si difficiles à mettre en oeuvre [23].

Dès lors, il faudra admettre de la souplesse et donc permettre l'incidence. A ce sujet, diverses études en psychologie cognitive ont montré qu'au fil des utilisations, l'utilisateur s'habituait au mode de travail de la machine et exprimait de plus en plus clairement ses idées, donc diminuait l'incidence [24].

La gestion du niveau d'incidence permis devra donc être essentiellement dynamique. Pour des actions sur la tâche, dont le système sait que l'utilisateur n'est plus néophyte, on sera beaucoup moins souple .

Idéalement donc, la gestion du dialogue en LN est un processus dynamique dont le but est de s'adapter au contenu informationnel des énoncés de l'utilisateur.

---

[23] Même lorsqu'on réduit le LN à un sous-langage....

[24] Ainsi, [Chantraine 90] note : "*Classiquement, on observe une baisse significative du temps et du nombre de tours de paroles (ainsi qu'une simplification de leur structure) nécessaire pour effectuer un rangement identique lorsque cette tâche est répétée plusieurs fois successivement avec les mêmes sujets.*"

Or, comme nous le verrons par la suite, la machine ne communique pas ou peu en LN avec l'utilisateur. Elle affichera des fenêtres, icônes, boutons, ...; produira des sons, ... Ces types d'interaction sont pour le moins directs. Comme on a vu précédemment que l'utilisateur adaptait sa façon de communiquer avec la machine à celle utilisée par la machine pour communiquer avec lui, on peut s'attendre à ce qu'il se rapproche des énoncés maximalises de Luzzati lorsqu'il utilisera le LN.

3.4. Modèle de la tâche et planification

Le rôle de l'interface est de régler le dialogue entre l'utilisateur et l'outil.

Pour ce faire, nous avons défini deux types de dialogues.

Le dialogue externe regroupe les échanges entre l'utilisateur et la machine. L'utilisateur produit des énoncés en LN finalisé (langage de commande) contenant des actes de langage de requête.

La reconnaissance de ces actes par l'interface fait en sorte que les intentions de l'utilisateur de modifier la tâche deviennent des intentions pour l'interface. L'interface détermine alors un **plan d'actions** adéquat.

Lorsque le processus de reconnaissance échoue à quelque niveau que ce soit (bruit, erreur de référence, intention non réalisable, ...), l'interface doit en avertir l'utilisateur. On entre alors dans un processus de négociation.

Le dialogue interne, pour notre application est asynchrone; en réalité il n'y a pas à proprement parler de dialogue : l'interface envoie des ordres à l'application et continue ensuite son travail. L'application se contente de signaler à l'interface qu'elle a fini en envoyant un code de retour (indiquant les identifiants des objets créés, ...)

Lorsqu'un ordre est adressé à l'outil par l'interface, il n'y a que très peu de possibilités de revenir en arrière si elle s'est trompée (un ordre inexécutable étant donné la situation peut tout simplement stopper le système). Dès lors, avant de lancer un ordre à l'outil, l'interface doit s'assurer du bien fondé de cet ordre. L'objectif est que **l'interface ne demande à l'application que des choses possibles.**

La figure 3.4 illustre le rôle pivot de **l'interface.**

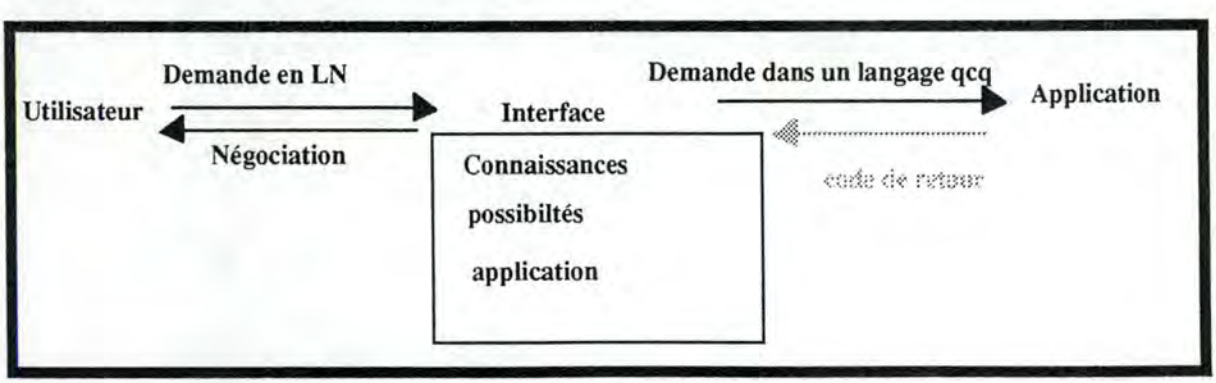


Figure 3.4 : Place de l'interface entre utilisateur et application.

Pour réaliser cet objectif, l'interface doit impérativement disposer d'une représentation de l'état de l'application et des différentes actions qu'elle comprend. On appelle ce modèle, le **modèle de la tâche** ou de l'application.

De plus, il faut que la représentation de ce modèle soit compatible avec la représentation de ses intentions sur la tâche.

Lorsque l'interface à reconnu la requête de l'utilisateur, elle a fait sienne ses intentions. Elle connaît l'état actuel de la tâche (le modèle de l'application n'est jamais en retard par rapport à la réalité de la tâche). Elle connaît également l'ensemble des actions qu'elle peut appliquer pour modifier la tâche. Elle va donc chercher un **plan d'actions** pour réaliser ces objectifs.

Si elle en trouve un, elle modifie son modèle de l'application (anticipation des résultats) et lance la séquence d'action. En retour, elle reçoit de l'application une validation de ses prévisions.

Si la planification échoue, c'est alors que l'énoncé n'était pas assez explicite que pour pouvoir être interprété, elle le signale à l'utilisateur et rentre alors dans un processus de négociation.

Ce double rôle de l'interface, valider les demandes et générer des ordres pour y répondre, est donc typiquement un problème de planification.

Tout ceci me permet de faire le lien avec le concept de monde réel présenté au chapitre 2. Le modèle de l'application de l'interface est donc bien une composante de ce monde réel. Cette représentation doit être la plus proche possible de la tâche. Grâce à la planification, on permet à l'interface d'anticiper l'état futur de la tâche; on assure donc cette symétrie (seul un code de retour est envoyé de l'outil à l'interface pour valider la représentation anticipée à la fin de la séquence d'actions).

La réalité de la tâche doit aussi être perçue par l'utilisateur (au minimum à intervalles). On suppose donc qu'il est tenu au courant de l'évolution de l'état des objets tout au long de son utilisation de l'outil. En principe, seule l'interface est en contact avec l'utilisateur. Elle doit donc remplir cette tâche le mieux possible. Nous verrons, au chapitre 6, différentes voies pour y arriver.

3.5. Dialogue dans Multiworks

3.5.1. Langage de commande

On utilise le langage pour préciser ses intentions jusqu'au moment où on reçoit entière satisfaction (Figure 3.5).

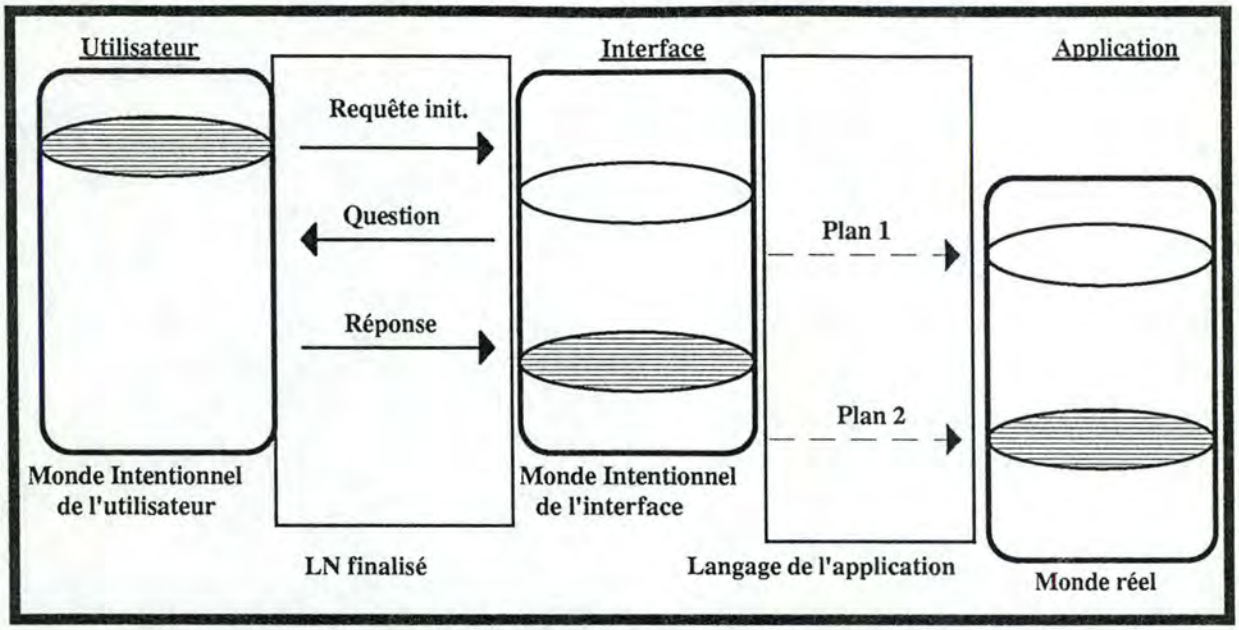


Figure 3.5 : Préciser ses intentions

La figure 3.5 retrace le mécanisme de transformation d'intentions en objets "réels" de l'application. L'utilisateur émet une requête : "je voudrais voir apparaître une ellipse rayée". Ellipse rayée fait jusqu'alors uniquement partie de son monde intentionnel. Cette proposition comporte un acte de langage (une requête) qui a comme conséquence d'inscrire une ellipse dans le monde intentionnel de l'interface.

Il faut croire que l'interprétation de cette proposition pose des problèmes à l'interface. Elle n'arrive pas à décider de la nature des rayures. Elle se rend compte de cette difficulté, et la rapporte à l'utilisateur en émettant une question. Simultanément, elle envoie un plan d'actions à l'application pour dessiner ellipse.

L'utilisateur répond à l'interface, celle-ci se satisfait de la réponse, modifie ses intentions et lance un deuxième plan d'action ad-hoc.

On aurait pu imaginer plusieurs variantes à cette situation (par exemple, que l'interface ne se rende pas compte de sa mauvaise interprétation, qu'elle se contente de faire afficher "*ellipse*" et que par conséquent, la modification de cette "mauvaise réponse" ne se fasse plus par une réponse à une question mais par une deuxième requête de l'utilisateur plus ou moins indépendante).

De façon générale, on peut considérer qu'une requête sera suivie, d'une part, d'une série d'actions sur le réel et, d'autre part, d'un échange (question-réponse) entre l'interface et l'utilisateur (Figure 3.6). Dans tous les cas, il faut que la négociation s'arrête; c'est à dire qu'a un certain moment, l'interface se contente d'agir sur le réel.

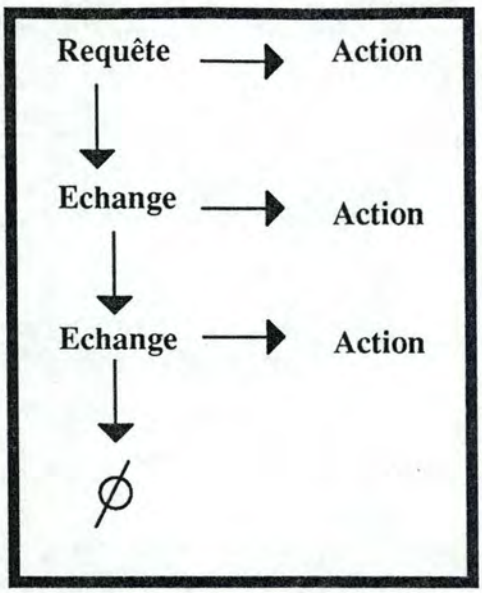


Figure 3.6.

Qu'en est-il exactement dans Multiworks ? Voici un exemple de dialogue supporté dès à présent par notre système :

Utilisateur : ouvre le noeud Mailbox

Machine : *action*

U : crées-y un bouton

Machine : *action*

U : mets ce bouton à droite du bouton Sauve

Machine : *action*

U : ouvre le fichier Essai

Machine : *action*

U : mets-le dans le script du bouton

Machine : *action*

U : ferme le noeud

Bien que l'objectif de l'équipe du CRIN n'était pas de développer un système de gestion de dialogue très bavard (mais plutôt de résoudre les problèmes de référence grâce au modèle des zones temporelles), on s'étonne tout de même du manque de réponses langagières de la machine.

Le processus représenté à la figure 3.6 est réduit à sa plus simple expression : l'utilisateur donne un ordre, la machine agit.

L'introduction de la planification à Multiworks devrait permettre d'augmenter le nombre et la complexité des ordres. On peut s'attendre, dès lors, à ce que la machine complexifie le dialogue.

Notre but est, avant tout, de fournir une réponse opérationnelle à une requête d'un utilisateur. On veut le plus possible se contenter de la relation : REQUETE -----> ACTION et ne pas rentrer dans des sous-dialogues d'éclaircissement des ordres ou des réponses.

La réalité étant tout autre[25], on doit au minimum, préparer des arguments, un diagnostic, un compte rendu de l'interprétation (de la planification) pour pouvoir ensuite entrer dans un dialogue incident si l'interprétation de la requête n'aboutissait pas directement à une action (plus généralement à un plan d'actions).

Notre travail devra donc préparer au mieux ces arguments. En fonction de l'état d'avancement dans le dialogue, on modulera ses choix de diagnostics pertinents[26].

3.5.2. Outil et utilisateur

En principe, l'outil et l'utilisateur ne doivent pas se rencontrer directement. Mais dans le projet qui nous occupe, l'application à interfacer (MULTICARD) est un éditeur d'hyper document (outil) couplé avec un gestionnaire de fenêtres. Lorsque MULTICARD déplace une fenêtre, il rentre en contact directement avec l'utilisateur.

3.5.3. Utilisateur et interface

L'utilisateur émet des requêtes en LN finalisé.

L'interface présente les résultat de l'évaluation de ces requêtes : état de la tâche modifié ou causes de l'échec. Le chapitre 6 sera consacré à ces questions.

3.5.4. Interface et outil

L'interface communique avec l'outil en lui envoyant des ordres dans une syntaxe définie. L'outil, dès qu'il a exécuté ces ordres, renvoie un code de retour (Ex : l'identifiant du noeud créé).

3.5. Conclusion

L'introduction de l'interface entre la tâche et l'utilisateur permet de séparer, à l'intérieur de la machine, les traitements liés à l'outil de ceux liés à la communication entre cet outil et l'extérieur.

L'utilisateur est en mesure de communiquer avec l'interface car il existe des connaissances communes sur le code, sur le monde et la situation (monde réel) (Figure 1.1).

---

[25] Qui peut se vanter de comprendre directement ce qu'on lui veut?

[26] Toute cette partie : choix de diagnostic pour expliquer l'échec n'est actuellement pas implémentée. Néanmoins, dans la dernière partie de ce mémoire, on présentera quelques considérations qui devraient déboucher sur une génération optimale de réponses (questions incidentes).

Il n'y a pas de négociation possible entre l'interface et la tâche. Avant que l'interface n'agisse sur la tâche, elle doit être certaine d'elle, sous peine de bloquer le système. La planification entre le monde réel (modèle de l'application) et le monde intentionnel (intentions sur la tâche transmises par l'utilisateur) de l'interface doit permettre soit de trouver une séquence d'actions qui réalise ces intentions, soit d'expliquer l'échec à l'utilisateur si elles ne sont pas réalisables.

Le chapitre suivant sera consacré à la présentation du formalisme utilisé pour implémenter ces notions dans le projet Multiworks.

Chapitre 4

4. Le Temps dans le dialogue

Concepts : **Temps, zones temporelles, actions, objets.**

Ce chapitre est consacré à la présentation du modèle des Zones Temporelles ([Romary 89]) utilisé pour représenter les objets et actions dans Multiworks.

Bien que ce modèle soit développé, en partie, pour la résolution des problèmes de référence, nous montrerons tout au long de ce chapitre qu'il est adapté (ou qu'il peut facilement s'adapter) au notions précédemment introduites : monde réel, monde intentionnel, requête, plan, ...

Sans vouloir refaire une théorie complète du temps dans le dialogue, montrons, d'une part, qu'il est indispensable, entre autres pour des problèmes de calcul de références ([Gaiffe 92], [Romary 90], [Gaiffe 90]), de représenter l'état, donc les objets de la tâche, de façon temporelle. D'autre part, démontrons qu'une action est, avant tout, un processus temporel de transformation d'objets; et enfin, que l'utilisation d'un formalisme commun de représentation des objets et des actions de la tâche, bien que certains n'y souscrivent pas ou pas entièrement ("*the formalism used should make a clear distinction between actions and objects*", [Carbonell 91]) et plus particulièrement, le formalisme introduit par [Romary 89] et développé par la suite, facilite grandement le travail de l'interface.

4.1. La description des états est d'ordre temporel

Nous avons appelé état ou état de la tâche, la représentation de l'ensemble des objets de la tâche. Un état est clairement une représentation donc un modèle statique. Livré à lui même, cet état ne bouge pas. Néanmoins, et il faut l'espérer, lorsqu'on utilise un outil, cet outil fait évoluer l'état (on crée un noeud dans un groupe, on bouge une fenêtre, ...). L'état passe donc par toute une série de valeurs : E1...Ei....En; ces valeurs étant reliées par des relations temporelles(précède, suit, ..) [27].

[27] Pour plus de détail sur les relations temporelles, cfr. infra.

4.1.1. Modèle de l'application

Le monde réel pour l'interface, c'est avant tout le modèle de l'application. Ce modèle de l'application est construit avec des zones temporelles.

4.1.2. Monde intentionnel de l'interface

Lorsque nous avons défini les actes de langage du dialogue de commande, nous avons utilisé les opérateurs de croyance censés modéliser le monde intentionnel de l'interface et de l'utilisateur (suivant que le premier argument de ce prédicat était "speaker" ou "hearer").

Nous avons vu qu'en réglant une fois pour toutes le sort des actes perlocutoires, en définissant **Cause-to-want** et **Convince**, l'interface peut se passer des opérateurs de croyance "de" l'utilisateur (de ce que croit l'utilisateur). Les intentions de l'utilisateur sur la tâche deviennent (par les actes de langage) des intentions pour l'interface de modifier la tâche; ce sont donc des anticipations sur le modèle de la tâche.

On confondra les deux mondes (réel et intentionnel) dans le même modèle, en prenant soin de noter d'une certaine façon les zones anticipatives (Cfr. zones hypothétiques et contextes).

4.1.3. Mondes de l'utilisateur

Savoir comment l'utilisateur représente le monde de la tâche n'a que peu d'importance; il (l'interface) faut seulement s'assurer que l'utilisateur dispose d'une représentation correcte de celui-ci (Cfr. chapitre 6).

Le monde intentionnel de l'utilisateur est a priori hors contrôle. Néanmoins, l'interface ne modifiera la tâche (réalisera les voeux de l'utilisateur) que pour autant que la planification réussisse. La présentation de l'échec et de ses causes à l'utilisateur devra avoir comme effet de ramener les souhaits de l'utilisateur sur la tâche à des choses plus réalisables.

4.2. La description des actions est d'ordre temporel

Nous avons vu qu'une action est un mécanisme qui fait évoluer l'état. Traditionnellement, une action est définie par trois champs : des pré-conditions, des post-conditions et un champ de décomposition de l'action en actions plus élémentaires et/ou une suite d'instructions directement exécutables par l'application.

La temporalité d'une action est directement inscrite dans ces composantes. Les pré-conditions sont, en effet, des conditions que doit respecter l'état (les objets de l'état sur lesquels porte l'action) pour que l'action soit applicable. Les post-conditions (traditionnellement décomposées en retraits et ajouts à l'état) sont des conditions que vérifie l'état après exécution de l'action.

Il est clair qu'une action ne modifie pas tout un état (ne s'applique pas à tous les objets de l'état) mais à un certain nombre d'objets particuliers. On matérialise cette restriction sur les objets de l'état en précisant les variables paramètres d'une action et en exprimant les pré- et post-conditions par rapport à ces variables.

Si on considère le temps i comme temps de l'exécution de l'action, on peut représenter son exécution par la figure 4.1.

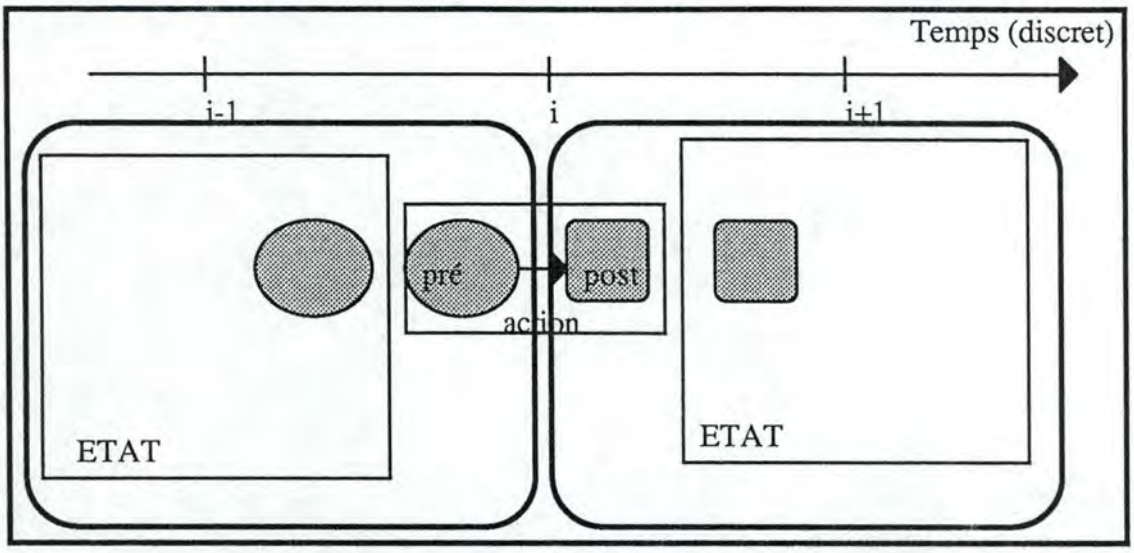


Figure 4.1: Temporalité d'une action.

En général, on considère le temps d'exécution de l'action comme nul : on se contente de simuler son exécution sur l'état, on n'exécute donc pas réellement l'action. On dira donc que l'état passe directement de Ei à Ei+1.

Cette dernière remarque nécessite quelques mots d'explication. J'ai dit précédemment combien il était important de mettre une interface entre l'application et l'utilisateur. Cette importance est d'autant plus grande que, dans notre cas, le logiciel d'hyper-texte avait été développé par des gens externes au CRIN; on ne savait donc pas le modifier directement. J'ai dit aussi qu'il était nécessaire que l'interface, en plus des connaissances permettant de traduire le LN finalisé en représentation interne, et cette représentation interne en commandes de l'hyper-texte, dispose d'un modèle de l'application, c'est-à-dire d'une représentation interne de l'état de l'application et des actions. Lorsque l'interface reçoit ordre de réaliser une action, elle vérifie sur sa représentation de l'état qu'elle est applicable; si oui, elle l'applique : instantanément, elle modifie sa représentation de l'état et envoie la commande à l'application. L'exécution de cette commande ne se fait pas instantanément (ex : émission d'un beep pendant 20 minutes). Il n'empêche que, dès qu'elle est physiquement terminée, l'application envoie un signal à l'interface qui lui permet de valider sa représentation anticipée de l'état.

A titre d'exemple, prenons l'action qui crée un noeud dans un groupe (action combien célèbre !). Cette action est modélisée par la figure 4.2.

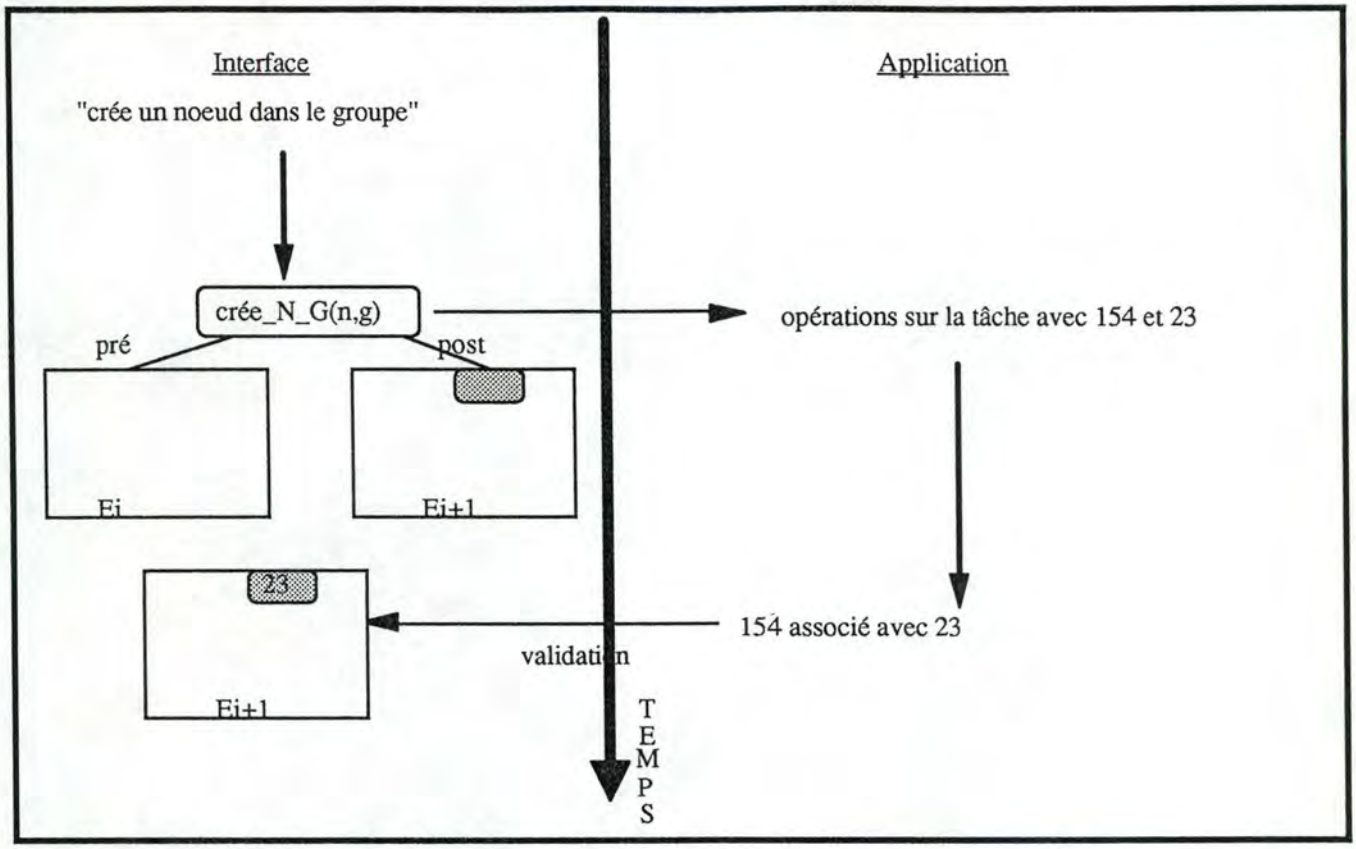


Figure 4.2 : Création d'un noeud dans un groupe.

L'action crée_N_G n'a pas de pré-conditions sur l'état de la tâche, elle peut donc être directement exécutée. Pour ce faire, l'interface envoie une demande de création d'un noeud à l'application (opérations sur la tâche avec 154 et 23) et modifie l'état de la tâche en ajoutant (retirant) les post-conditions.

Lorsque l'application en a terminé avec ces opérations sur la tâche, elle envoie un code de retour à l'interface, pour lui signaler quelle a fini et que le noeud porte l'identifiant 23, validant par la même occasion, la modélisation anticipée de la tâche.

Dès lors que l'exécution d'une action (en simulation) n'a pas de durée, on se centrera sur un modèle de représentation discrète du temps et sur les relations d'ordre entre faits.

4.3. Un modèle de représentation du temps

Les concepts de base de ce modèle ont été établis par [Romary 89] et développés dans les articles qui ont suivi.

4.3.1. Concepts de base.

Comme le note [Gaiffe 92], "*Dans ce modèle, la brique de base est la zone temporelle. Une zone temporelle contient des faits et entretient des relations avec d'autres zones*

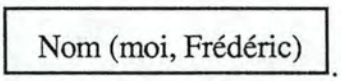
temporelles. Ces relations sont : l'inclusion (....) et l'adjacence.". On fait difficilement plus simple.

A ces concepts, on fait correspondre un formalisme de représentation graphique.

Zones temporelles

La zone temporelle définit un intervalle temporel de validité d'un fait. On utilise la logique des prédicats du premier ordre pour exprimer ce fait.

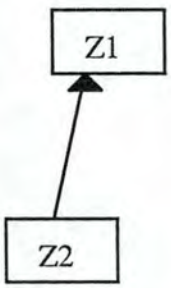
Ainsi, la zone temporelle associée au fait "Je m'appelle Frédéric" [28] sera représentée par :



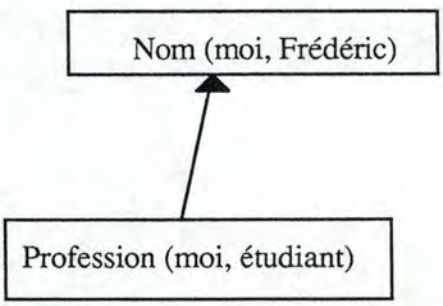
La temporalité du modèle apparaît lorsqu'on relie ces zones entre elles.

Inclusion

Si une Zone Z2 est incluse dans Z1 (Z1 contient Z2), cela implique que Z2 n'est valide que pour autant que Z1 le soit.



Z2 exprime un sous fait de Z1. Par exemple, on peut représenter " je m'appelle Frédéric et je suis étudiant " par le système :



Ici, on a supposé que je conserverai le même prénom tout au long de ma vie [29], et que par conséquent, être étudiant est bien un sous fait de mon identité. Est-ce toujours le cas? On

---

[28] Pour bien montrer que la syntaxe des zones temporelles peut s'appliquer à n'importe quel contenu informationnel, nous avons pris, pour la présenter, des exemples quelconques.

[29] Au moins tant que je suis étudiant.

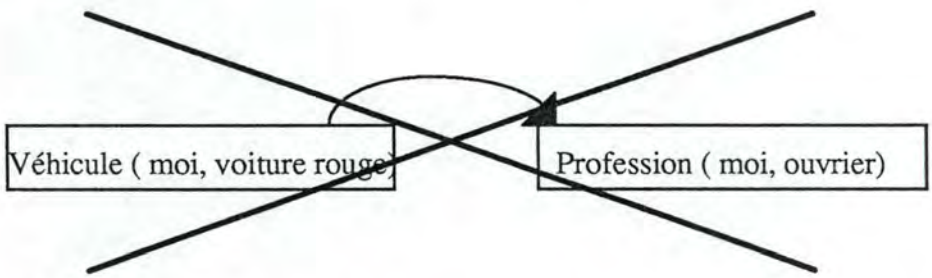
voit ici les problèmes d'arbitrage que peut rencontrer le modélisateur du système. Qu'en est-il si mon prénom change ? (Problèmes au niveau de la sémantique.)

Adjacence

La relation qui exprime précisément l'évolution des zones (relation de précédence et de succession) est la relation d'adjacence. Z2 suit Z1 signifie que les faits contenus dans Z2 prennent le pas sur ceux contenus dans Z1 (faits et sous-faits).

Il faut donc en principe que ces faits soient sur une même ligne d'évolution : on pourra par exemple faire suivre le fait que je suis ouvrier avec celui d'être étudiant; on voit mal comment on pourrait lui faire précéder le fait que je possède une voiture rouge. (Encore un problème sémantique)

et pas :

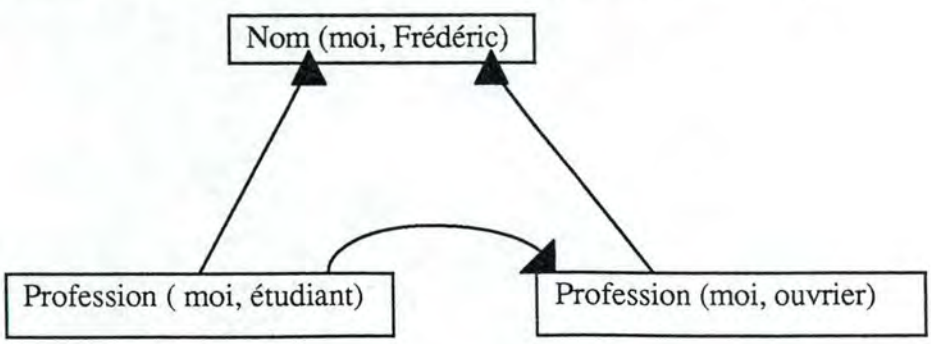


Il faut que les faits soient disjoints (et donc sur une même ligne de propriété, Cfr. infra) pour que les zones qui les reprennent soient reliées par une relation d'adjacence.

Combinaisons

Deux zones ne peuvent jamais être reliées simultanément par une relation d'inclusion et une d'adjacence. L'adjacence exprime un changement d'intervalle temporel de validité tandis que l'inclusion est une relation qui fait hériter les sous-faits des propriétés de la zone mère et donc est incluse dans le même intervalle de validité.

On peut néanmoins utiliser ces deux relations indépendamment. Ainsi, "je m'appelle Frédéric, j'étais étudiant mais depuis peu, je suis ouvrier" peut être modélisé par :



Le sous-fait validé, vu l'état du système, est celui qui se trouve le plus à droite de la chaîne pour autant que son "fait père" soit encore valide (soit lui-même le plus à droite et fils d'un père valide, ...)[30].

Combiner les zones temporelles entre elles, permet donc de retracer toute l'évolution des attributs d'un objet en utilisant une syntaxe triviale.

4.3.2. Définition formelle

On trouve dans [Romary 91], une définition plus formelle de la syntaxe. Les règles de construction sémantiques (contenu des zones temporelles), dépendent bien évidemment de l'utilisation qu'on fera de cette syntaxe.

Formalisme

Commençons par décrire la typologie employée. Soit A, un ensemble d'objets, et R_A, un ensemble de relations sur ces objets. On appelle **fait**, un terme composé d'un élément de R_A appliqué à des éléments de A (le nombre d'éléments dépend de l'arité de la relation).

On représente le monde (réel et intentionnel) au moyen d'un ensemble de faits que l'on considère comme valides à ce moment de la représentation.

On utilise le formalisme : $\frac{f_1, f_2, \ldots, f_n}{f_{n+1}}$ pour signifier que le fait f_{n+1} est inférable des faits $f_1, \ldots f_n$.

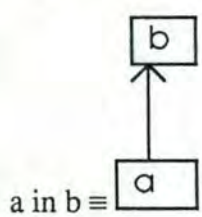
On indique que des faits sont incompatibles les uns avec les autres (que de ces faits, on ne sait inférer aucun autre fait) par : $\frac{f_1, \ldots, f_n}{\Diamond}$.

Définition

Soit Z l'ensemble des zones temporelles.

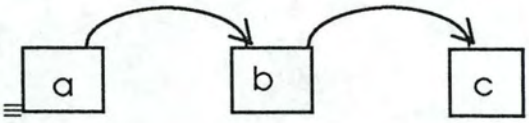
Soit R_Z = {in, prec}, l'ensemble des relations sur Z. Ces relations sont d'arité 2.

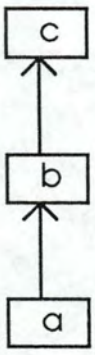
$\forall a,b,c \in Z$:

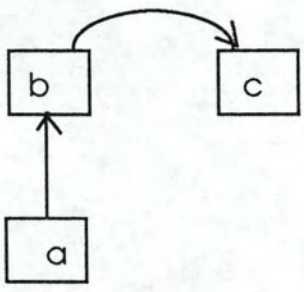


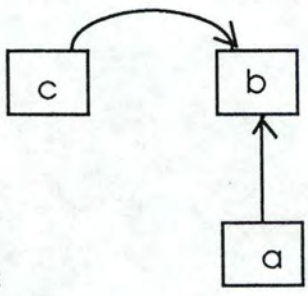
[30] Par cette dernière remarque, on se convainc de la nécessité de développer un algorithme qui, partant de l'historique (ensemble des zones temporelles des différents objets) extraira les zones valides au temps de l'exécution.

a prec b ≡ 

$$\frac{a \text{ prec } b,\ b \text{ prec } c}{a \text{ prec } c} \equiv$$ 

$$\frac{a \text{ in } b\ ,\ b \text{ in } c}{a \text{ in } c} \equiv$$ 

$$\frac{a \text{ in } b\ ,\ b \text{ prec } c}{a \text{ prec } c} \equiv$$ 

$$\frac{a \text{ in } b\ ,\ c \text{ prec } b}{c \text{ prec } a} \equiv$$ 

$$\frac{a \text{ in } b\ ,\ b \text{ in } a}{\Diamond} \text{ et } \frac{a \text{ prec } b,\ b \text{ prec } a}{\Diamond}$$

$$\frac{a \text{ in } b\ ,\ b \text{ prec } a}{\Diamond} \text{ et } \frac{a \text{ in } b\ ,\ a \text{ prec } b}{\Diamond}$$

4.3.3. Evolution du modèle

Les concepts ajoutés au modèle de base sont : les lignes de propriétés, zones hypothétiques, contextes, ... Nous les décrirons en détails dès que nécessaire.

4.4. Modélisation de l'état de la tâche [31]

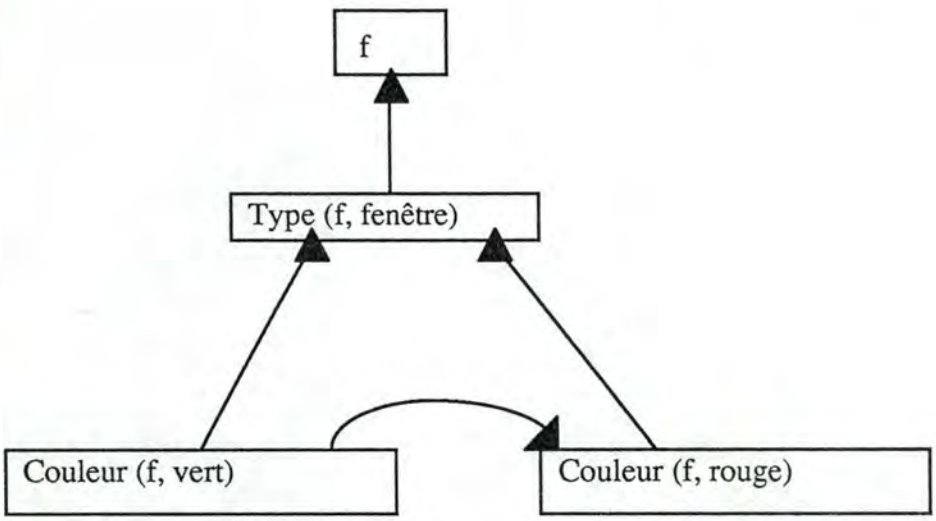
4.4.1. Modélisation des objets

Un objet de la tâche est modélisé par un graphe de zones temporelles.

Identification

La zone sommet de cet arbre est l'identifiant de cet objet. Dès que l'on crée un objet, MULTICARD va lui attribuer une valeur qu'il conservera et qui l'identifiera tout au long de son existence.

Typiquement, dans l'exemple suivant, l'identifiant de la fenêtre est **f.** Je représente donc l'objet **f** par :



L'identifiant d'un objet sera repris tel quel dans une zone, indépendamment de tout prédicat.

Lignes de propriétés

En ne se référant qu'à la syntaxe des zones (des faits inclus dans les zones), il est impossible de savoir si une nouvelle zone va suivre, être incluse ou restera indépendante d'une autre zone. Comme le note [Gaiffe 92], c'est typiquement un problème de consistance du graphe : "*Ainsi, si une zone Z1 contient un ensemble de faits F et qu'une zone Z2 contient un ensemble de faits G et que F union G est un ensemble inconsistant, alors on a forcément soit Z1 précède Z2, soit Z2 précède Z1 mais en aucun cas Z2 incluse dans Z1, ni Z1 incluse dans Z2. Sinon, l'ensemble du graphe temporel est inconsistant....le test de la consistance d'un ensemble de formules est indécidable dans le cas général*".

Pour lever cette inconsistance, à coté du graphe lui-même, on définit des lignes de propriétés, c'est à dire des ensembles de valeurs qui nous semblent opposés mais qui sont

[31] A partir d'ici, nous reprendrons des exemples propres à Multiworks.

sémantiquement liés. Par exemple, le fait d'être de couleur rouge doit appartenir à la même ligne de propriétés que celui d'être de couleur verte.

Lorsqu'on ajoute une nouvelle zone, on la fera succéder à la dernière zone valide qui appartient à la même ligne de propriété; s'il n'y en a pas, on l'inclura seulement à la zone père.

Ces lignes de propriétés sont hiérarchisées, ce qui permet de résoudre les problèmes d'inclusion.

En général, une ligne de propriétés sera définie par un même nom de prédicat .

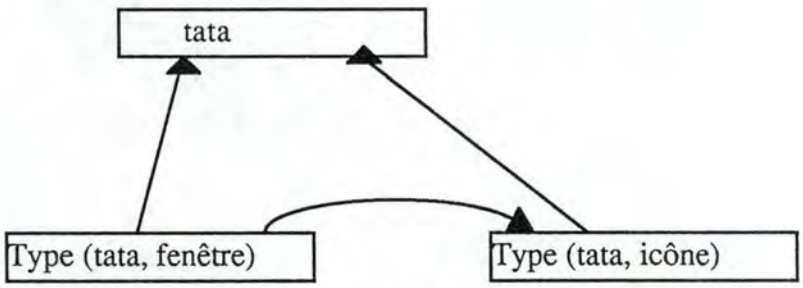
La ligne de propriétés de base du système est celle qui reprend l'identifiant des objets.

Une ligne de propriété particulière : le type de l'objet

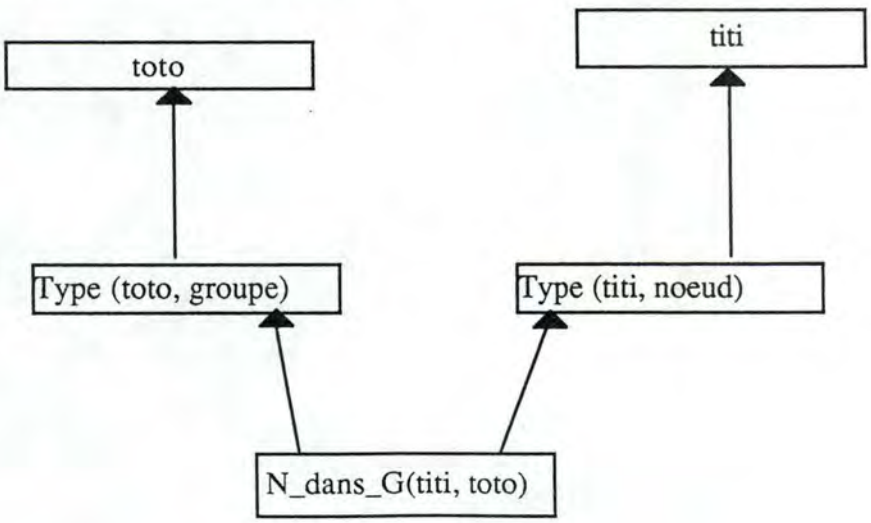
Un objet peut voir son type changer au cours de l'application : on peut par exemple icônifier une fenêtre. Le type est clairement une ligne de propriétés. Il est fondamental, car la première chose que l'on vérifiera lorsqu'on exécutera une action, c'est bien lui. Nous le placerons comme le seul fils de l'identifiant. Le reste des prédicats caractérisant l'objet lui seront reliés plus ou moins directement.

Quelques exemples

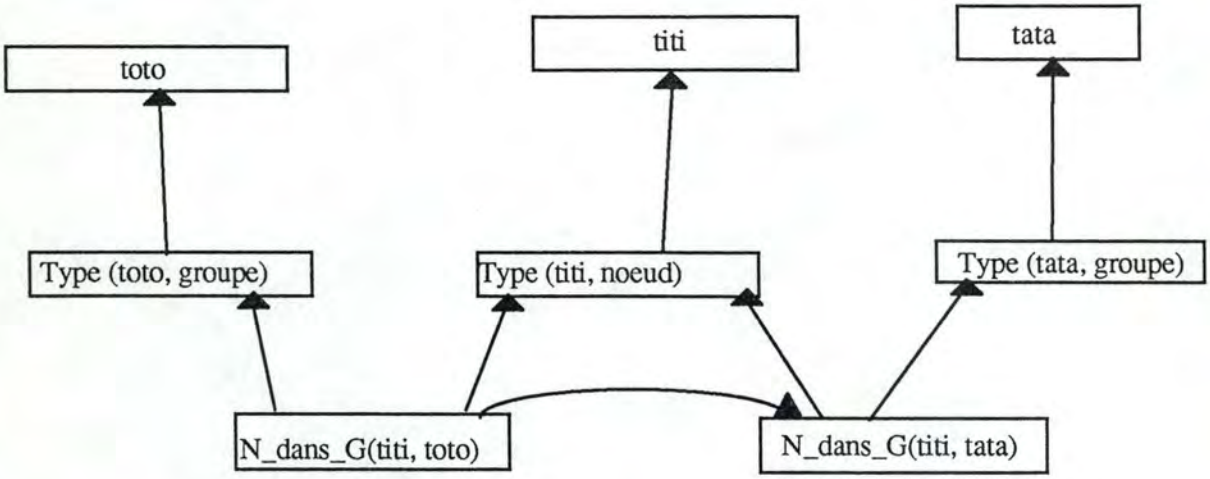
Un objet de type fenêtre devient une icône :



Il est clair que les objets ne sont pas séparés, il existe des relations qui les unissent. Ainsi, un groupe est composé d'un certain nombre de noeuds :

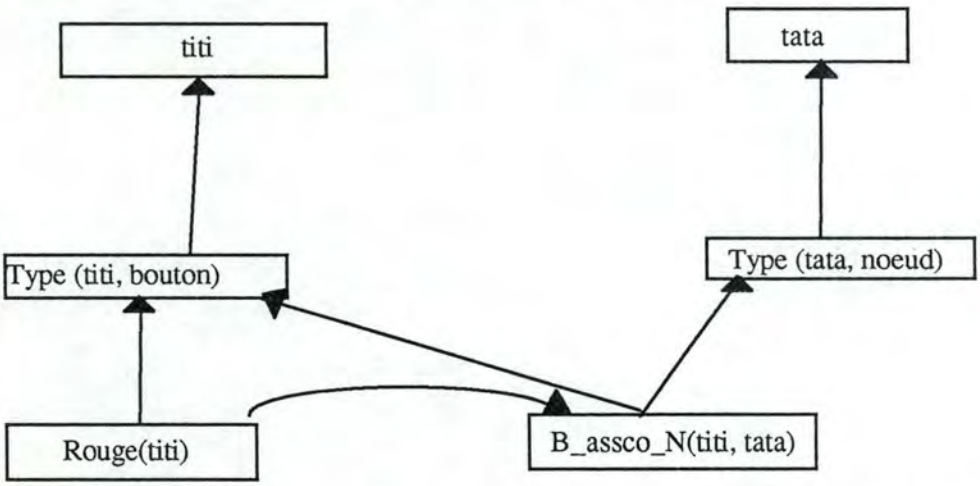


On peut arriver, en combinant ces relations, à des représentations de choses assez complexes : ainsi, le fait que le noeud titi a fait partie du groupe toto mais que maintenant, il est dans le groupe tata [32]:



Dans le dernier exemple, on retrouve trois lignes de propriétés : l'identifiant, le type et la relation "noeud dans groupe". A cette dernière, on ne fera succéder que des relations "noeud dans groupe" .

Dans l'exemple suivant : le bouton tata est rouge tant qu'il n'est pas relié à un noeud. On le relie au noeud titi, on oppose le fait que le bouton soit rouge : Rouge (titi) avec celui qu'il soit relié à un noeud : B_assoc_N(tata, titi); ces deux faits doivent donc être sur la même ligne de propriété (Cfr. le problème de la sémantique associée au modèle).



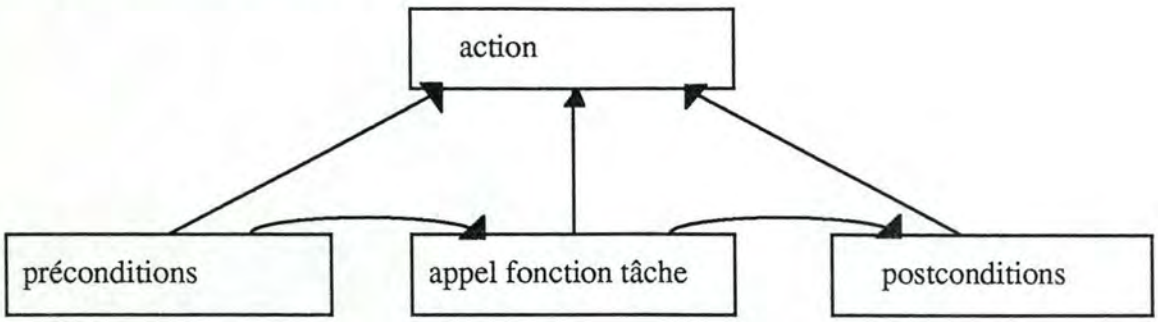
Le système défini par l'ensemble de ces zones et de leurs relations détermine bien ce que d'aucuns appellent **historique du dialogue** ([Pierrel 87], [Roussanaly 88]).

4.4.2. Modélisation des actions

[32]On utilisera titi, tutu, toto, tata, ... comme identifiants.

Structure de base

On représente une action par une structure de la forme :



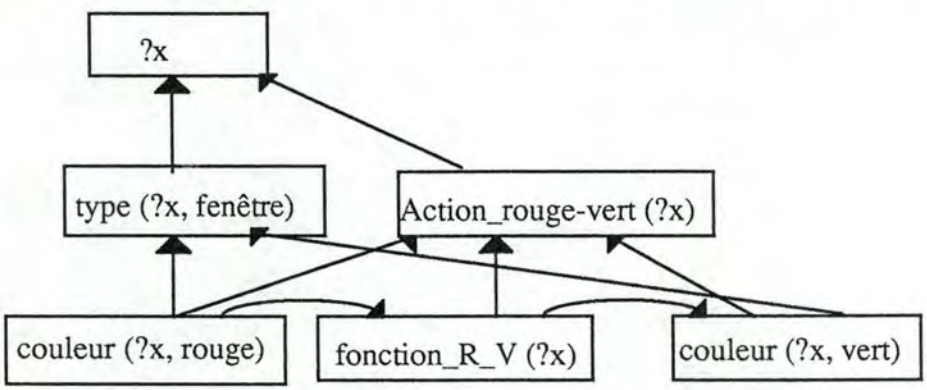
Une action est une structure temporelle composée d'une vérification de pré-conditions suivie, si ces pré-conditions sont respectées, d'un appel à une fonction de l'application, et de l'ajout de post-conditions aux objets de l'état.

On n'attend pas que la fonction de la tâche soit terminée pour imputer les post conditions au modèle. Quand cette exécution est complétée, on valide alors le modèle.

Les pré-conditions sont un filtre sur les objets de l'état. Elles servent, entre autres, à instancier les variables de l'action[33] . Les variables restantes (données créées ou mises à jour par l'application) sont instanciées par l'application, qui envoie un code de retour à l'interface dès que la fonction de la tâche est terminée.

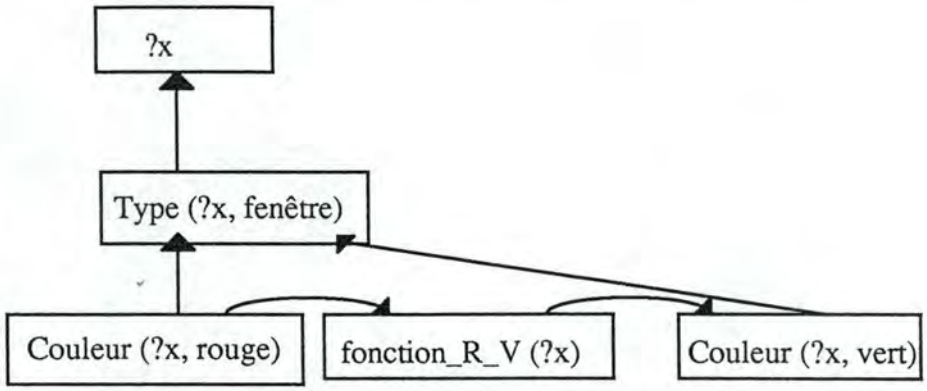
Jusqu'à présent, nous avons réduit la partie "décomposition" d'une action à un simple appel à une fonction de la tâche. Nous nous étendrons plus longuement sur cette partie dans le chapitre 6 lorsque nous parlerons d'agrégation d'actions et de rhetorical acts.

A titre d'exemple, on définit la fonction qui met une fenêtre rouge en vert comme suit :



que l'on représentera plus volontiers par :

[33] Une variable est représentée par **?lettre**.

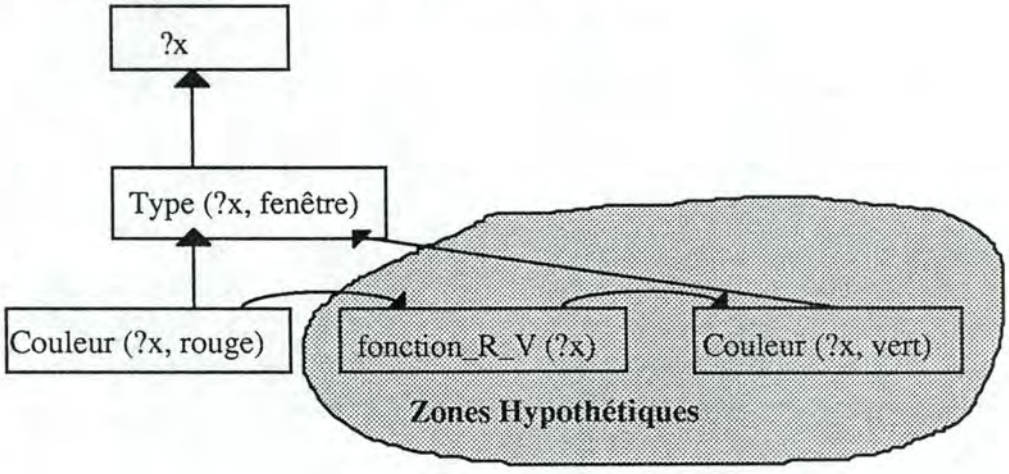


Les zones hypothétiques

Puisqu'il n'y a pas synchronisation entre l'application des post-conditions et la fonction de la tâche, ni entre l'appel à la fonction de la tâche et l'exécution de cette dernière, lorsqu'on applique une action à un objet, on est obligé de marquer les zones de post-conditions et d'appel à la fonction de la tâche d'une certaine manière, ce qui nous permet de distinguer dans notre modèle ce qui est sûr (ce dont on a reçu confirmation de l'application) et ce qui est encore hypothétique. On lève cette hypothèse dès qu'on reçoit confirmation de la fin et de la correcte exécution par l'application de la fonction de la tâche (+ instanciation des variables restantes).

Le passage de l'état hypothétique à celui de non hypothétique (réel) d'une zone, constitue la transition de valeurs du monde intentionnel au monde réel (Cfr. Figure 2.1).

On représente des zones hypothétiques de cette façon :



Lorsqu'on effectuera une recherche de plans, on n'exécutera pas les actions, les zones hypothétiques joueront donc un rôle fondamental.

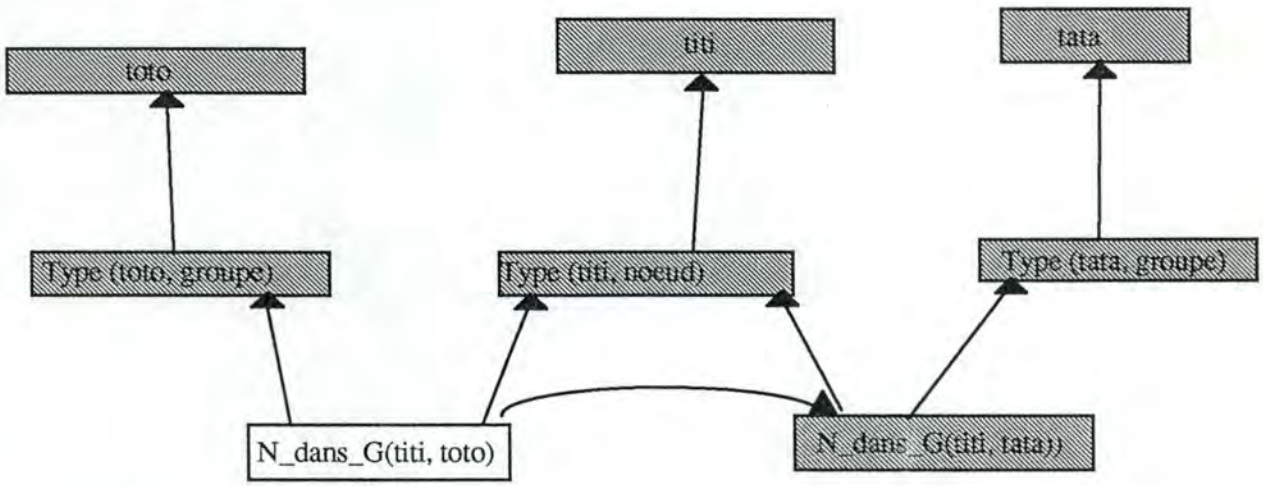
En introduisant les contextes au modèle, on permettra une gradation dans l'hypothétique.

Description de la fonction Recherche_nouveauté[34]

Au fur et à mesure que les objets de la tâche évoluent, on ajoute des zones à ces objets par la droite.

La fonction de Recherche_nouveauté est une fonction de l'interface qui agit directement sur le modèle de l'application. Le but de cette fonction est de déterminer quelles sont les informations nouvelles pour un objet donné.

Partant d'un ensemble de zones représentant un objet, j'ai dit que les zones valides (au temps T) sont les zones les plus à droites de la chaîne d'adjacence dont le père est également valide. Les zones reprenant les identifiants sont toutes supposées valides. Ainsi, sur l'exemple suivant, les zones valides sont hachurées :



On détermine ainsi l'information contemporaine contenue dans le modèle.

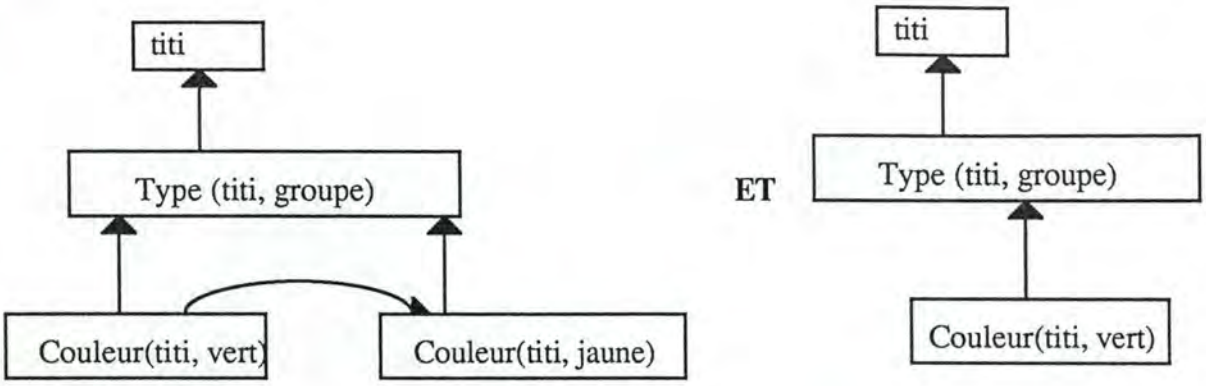
Pour vérifier si une action est applicable étant donnée l'état courant du modèle, je dois le comparer avec les pré-conditions de l'action. Si les pré-conditions n'apportent rien de nouveau, alors, je peux appliquer l'action. Si par contre, les pré-conditions apportent quelque chose de neuf au modèle, je ne pourrai exécuter l'action tant que je n'aurai pas ajouté ces nouveaux éléments à mon modèle.

La fonction recherche-nouveauté retourne à partir de deux ensemble de zones, un ensemble d'inclusions-adjacences qu'il faudrait ajouter au premier ensemble pour que le deuxième n'ait rien de nouveau par rapport au premier.

Exemple :

---

[34] Signalons à toutes fins utiles que les fonctions présentées et les autres ont été développées en LISP. Notre travail consistait à reprendre les fonctions déjà existantes (développées principalement pour la recherche des référents), à les adapter et à en créer de nouvelles qui permettent la planification.



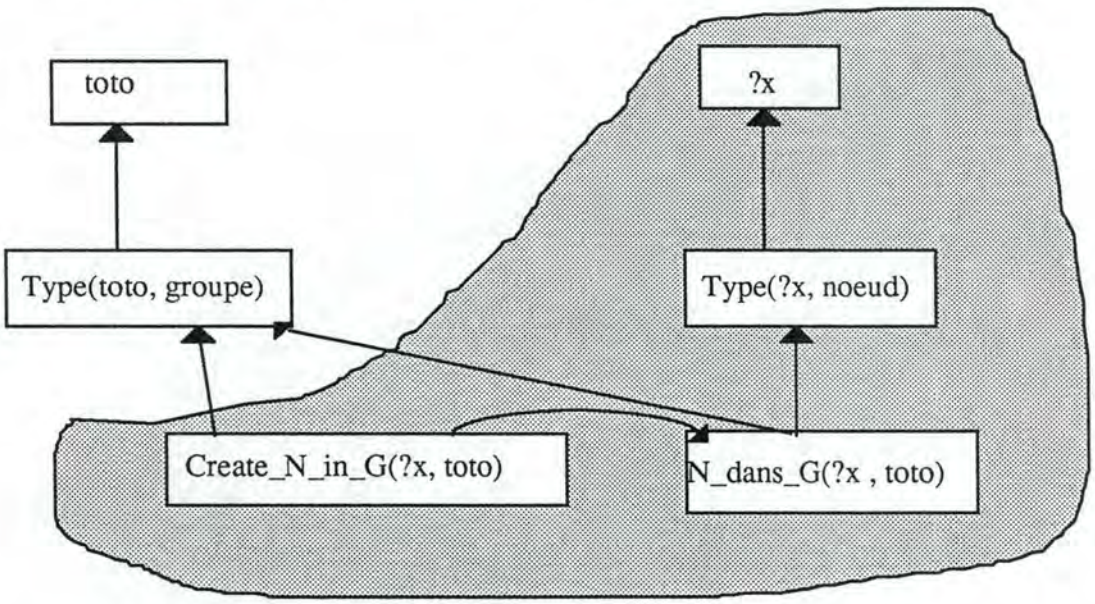
retourne : **Adjacence entre Couleur(titi, jaune) et Couleur(titi, vert).**

Cette fonction permet donc de vérifier qu'une action est applicable à un état et, dans la négative, de déterminer un objectif à réaliser pour que cette action soit exécutable.

Exécution d'une action dans l'interface

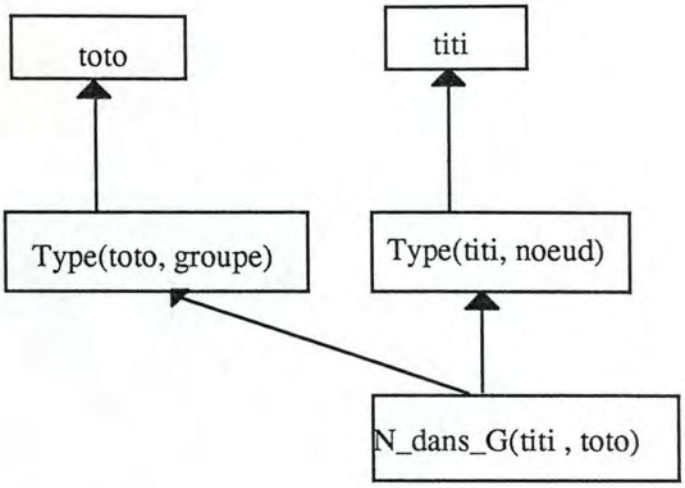
Lorsqu'on veut appliquer une action aux objets, il suffit de substituer les variables de cette action avec les identifiants de ces objets et de faire suivre les zones des objets qui vont être modifiées par l'action instanciée, pour autant que les pré-conditions soient vérifiées.

Soit par exemple l'action Create_N_in_G (?n, ?g) à appliquer à l'objet existant toto.



L'action Create_N_in_G est applicable à l'objet toto car la seule pré-condition de cette action porte sur le type de l'objet qui doit être un groupe. L'action et l'ensemble des zones des post-conditions sont jusqu'alors hypothétiques; aucune action de la tâche n'a été appelée.

L'interface demande alors à la tâche de lancer l'action (ou l'ensemble des actions) proprement dite. La tâche signale à l'interface qu'elle a fini en renvoyant un code de retour. Elle signale qu'il y a un nouvel objet qui vient d'être créé, par exemple titi, qui est de type noeud et qui est relié au groupe toto. L'interface s'attendant à une telle réponse, décide de substituer titi à ?x , et sort ces zones de la partie hypothétique. La zone contenant l'action est éliminée.



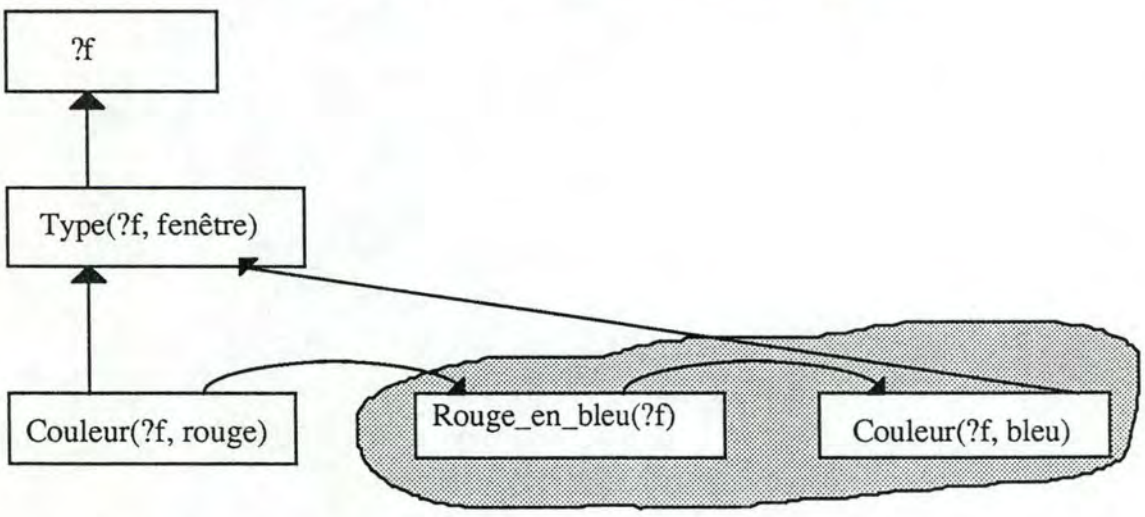
Quelques exemples.

Classiquement, on distingue trois type d'action : les actions de modification, de création et de suppression.

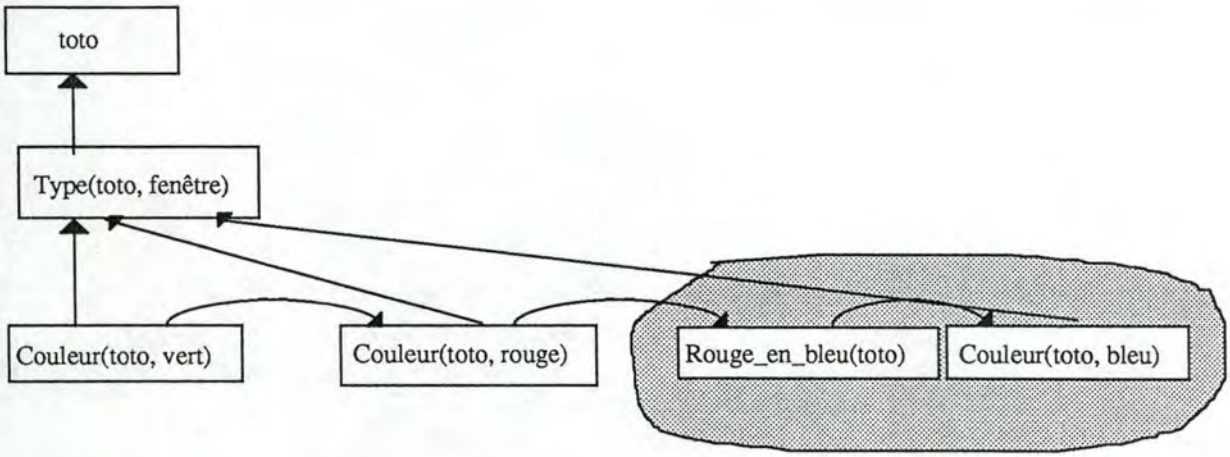
Modification

La modification est l'action la mieux maîtrisée, c'est aussi celle pour laquelle le modèle présenté est le mieux adapté car elle exprime une transition d'un état vers un autre. Avant d'appliquer une telle action, il est nécessaire d'en connaître tous les paramètres; l'objet modifié en est un.

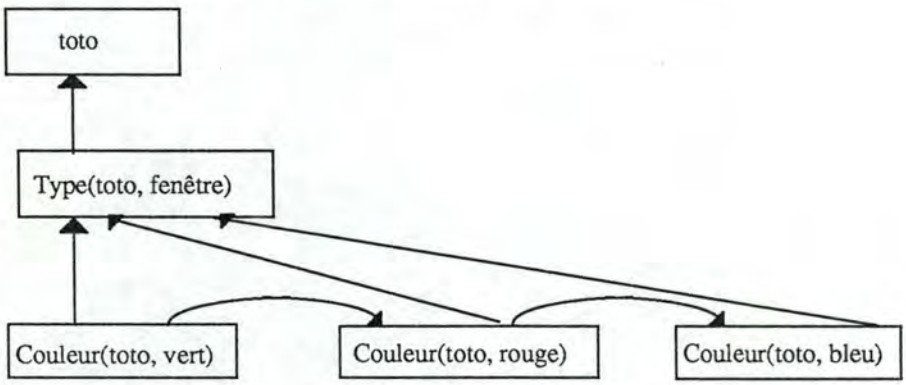
Soit donc l'opération Rouge_en_bleu, qui colorie une fenêtre rouge en bleu :



Appliquons cette dernière à la fenêtre toto qui était verte mais qui est maintenant rouge :



On obtient donc après réception de la validation de la tâche :



Création

On peut reprendre par exemple l'action Create_N_in_G. Ce type d'action a, par définition, peu ou pas de pré-conditions; elles sont donc toujours exécutables indépendamment de l'état de la tâche. Cela posera de gros problèmes lorsqu'on abordera la planification.

Suppression

Par suppression, on entend suppression de zones. Or, le principe même du modèle est qu'on ne supprime jamais de l'information. Ce n'est pas parce qu'un objet a disparu de la tâche qu'on ne peut le référencer. Par exemple, je peux très bien vouloir colorier une fenêtre en utilisant la couleur d'une autre que j'ai supprimée hier.

Supprimer un objet reviendra[35] donc tout au plus à le rendre inactif (par exemple en donnant une certaine valeur à son type).

4.5. Actes de langage et zones temporelles

4.5.1. De l'utilisateur à l'interface

Les actes de langage ont été présentés dans une optique de reconnaissance par l'interface des actes contenus dans le discours de l'utilisateur. Examinons ce que cela implique.

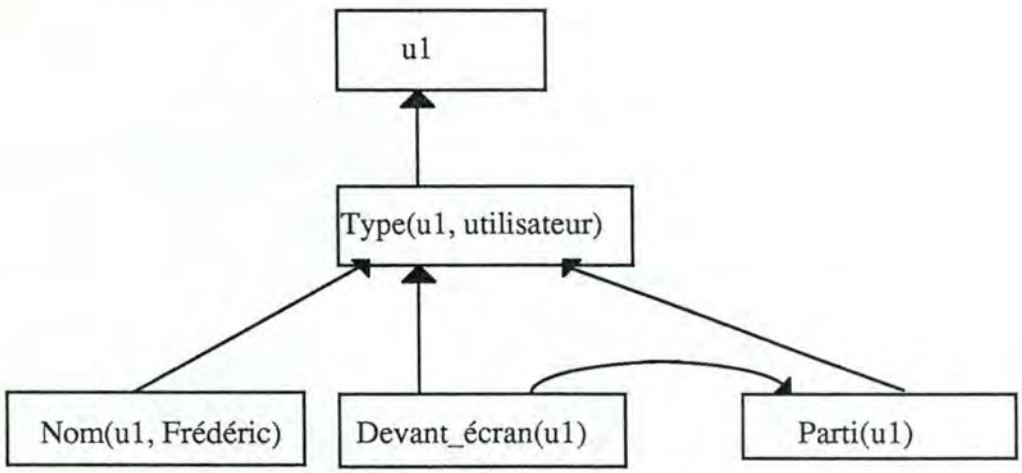
[35] Jusqu'à présent, ce type de problème n'a pas encore reçu de solution définitive satisfaisante.

Inform

Toute l'information que l'interface reçoit de la tâche est vraie. C'est d'ailleurs une pré-condition essentielle à la tenue d'un modèle de la tâche cohérent (Cfr. principe de coopération de Grice).

J'ai supposé jusqu'alors que le monde réel se limitait pour l'interface à ce modèle de la tâche. Néanmoins, on peut facilement imaginer que l'interface veuille étendre cette représentation de la réalité aux données concernant l'utilisateur; elle "voudrait" peut-être connaître son nom, son age, savoir s'il est actuellement devant son écran, ...

On peut facilement imaginer à côté du modèle de l'application des zones formant une structure de la forme :



qui renseignerait sur l'identité des utilisateurs, leurs habitudes, ... s'ils sont actuellement devant l'écran, ...

On verra également lorsqu'on parlera de la génération de réponses (Chapitre 6), l'importance de zones attachées aux zones du modèle de l'application et qui indiquent si l'utilisateur est en état de connaître ces faits (si sa représentation du monde réel comprend ces informations).

Lorsque l'utilisateur informe l'interface, il lui donne de l'information nouvelle ou actualise des données qu'elle avait déjà eu à connaissance. L'ajout ou la modification d'information se fait, dans notre modèle, par l'ajout de nouvelles zones en prenant soin de les relier aux anciennes en respectant les lignes de propriétés et autres règles sémantiques de la modélisation. Les faits ajoutés sont en principe vrais; ils seront donc contenus dans des zones réelles.

Request

Un request a comme conséquences d'ajouter des zones hypothétiques (à contexte infini) au modèle de la tâche et de lancer la planification avec ces zones comme objectif. Nous préciserons ce point au chapitre suivant.

4.5.2. De l'interface à l'utilisateur

Le chapitre 6 sera consacré en partie à régler cette question.

4.6. Conclusion

Les zones temporelles et la sémantique associée permettent de représenter le modèle de l'application dans l'interface. A cette représentation de la tâche, nous ajoutons des données sur l'utilisateur et élargissons par la même occasion le monde réel de l'interface.

L'interface peut anticiper les modifications de la tâche, déterminer un monde intentionnel lié à la tâche. Ces anticipations ne sont validées (mise dans le monde réel) qu'à partir du moment où la tâche les confirme.

De la même manière, l'ajout ou la modification d'information concernant l'utilisateur dans l'interface n'est possible que pour autant qu'elle ait reçu un acte d'information valide de celui-ci.

Une requête de l'utilisateur vers l'interface ajoute des zones au monde intentionnel de l'interface. Le chapitre suivant, consacré à la planification, doit permettre de faire le lien entre les intentions de l'interface, la découverte d'un plan ou gestion de l'échec et la validation par la tâche des anticipations de l'interface.

Chapitre 5

5.La planification dans le dialogue

Concepts : **planification, contexte.**

Le but de ce chapitre n'est pas de refaire une théorie complète de la planification, ni même de porter un jugement sur les méthodes traditionnelles de planification; il s'agit simplement de montrer en quoi la planification peut être utile dans un système de gestion de dialogue homme-machine.

Dans un premier point, nous exposerons brièvement les concepts et techniques classiques utilisées en planification, nous définirons ce qu'est un plan et verrons en quoi la planification peut être perçue comme processus d'apprentissage.

La deuxième partie du chapitre sera consacrée à relier les idées de la première avec le modèle des zones temporelles utilisé dans Multiworks. Nous y ajouterons les **contextes** et distinguerons les requêtes centrées sur l'action (obligation de moyen) de celles centrées sur le résultat (obligation de résultat).

Nous terminerons en présentant le planificateur que nous avons été amené à développer au CRIN.

5.1. La planification

Les différents concepts et techniques tentent de synthétiser [Allen 80], [Charniak 76], [Litman 87].

5.1.1. Concepts

Un **plan** est une séquence ordonnée d'opérations exécutables .

Une **opération** (ou action, ...) est un **plan** ou une **opération élémentaire** (opération directement exécutable par l'application).

Cette définition (récursive) traduit immanquablement une structure d'arbre. Une opération se compose, comme je l'ai déjà signalé de trois champs : pré-conditions, décomposition, post-conditions. Un plan étant une opération, il reprendra aussi ces trois champs. Nous nous intéresserons plus particulièrement à la partie décomposition.

Un exemple de plan tiré de [Litman 87] se trouve à la figure 5.1 [36].

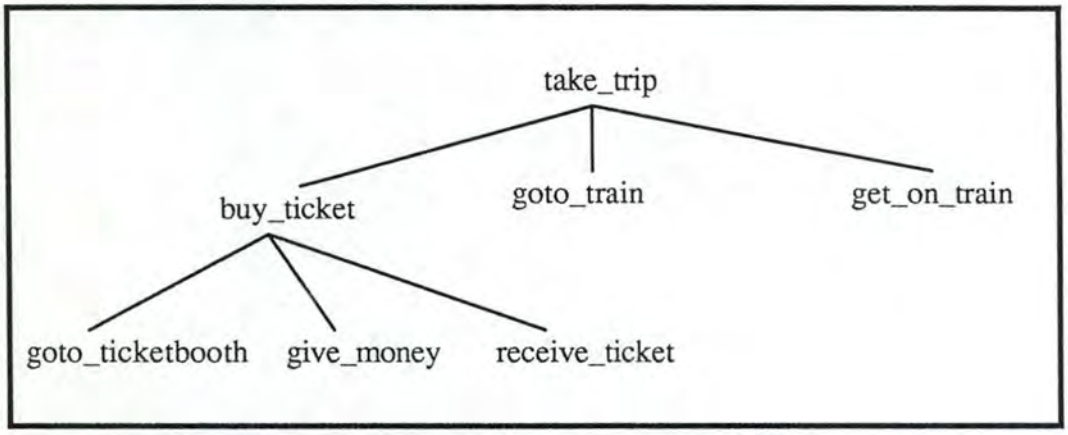


Figure 5.1 : Sketch of plan to take trip.

Nous avons déjà précisé ce que nous entendions par état (Cfr. chapitre 2). Lorsqu'on parle de construction ou de reconnaissance de plan, on entend trouver une séquence d'opérations qui, exécutées l'une à la suite de l'autre, relient un **état initia**l (réel) à un **état final** (intentionnel).

On appelle objectif, la différence entre les états initiaux et finaux.

Notre problème est le suivant : disposant, d'un ensemble d'opérations, d'un état de la tâche actuel, initial Ei et d'un état attendu, final de la tâche Ef, il faut trouver, créer un plan qui permette de passe de Ei à Ef.

5.1.2. Techniques

On présente ici les techniques de construction de plan les plus connues. En aucun cas on n'essayera d'optimiser la planification, on se contentera donc de ces techniques

Chaînage avant

Chaînage avant : on part de Ei qu'on modifie pour aller vers Ef (figure 5.2). L'état final n'est jamais modifié; dès qu'un Ei_n est égal à Ef, on remonte l'arbre jusqu'à Ei et on a le plan. La vérification de l'applicabilité des opérations sur les objets de l'état se fait sur les pré-conditions.

---

[36] Sémantique : les fils d'un noeud se succèdent dans le temps.

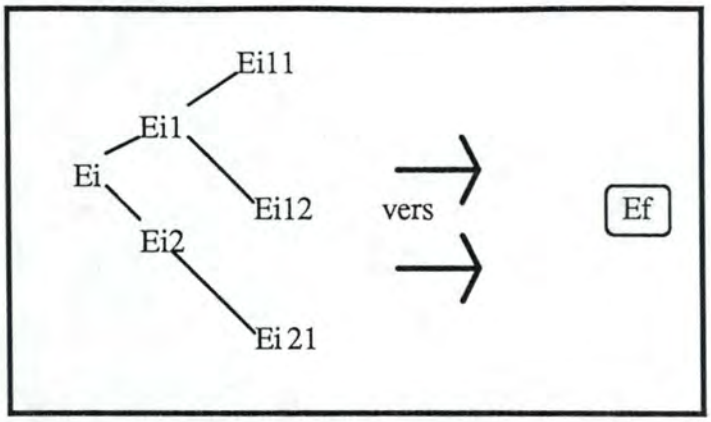


Figure 5.2 : Chaînage avant.

Chaînage arrière

Chaînage arrière : on part de Ef qu'on modifie pour aller vers Ei (figure 5.3). Ici, c'est Ei qui ne bouge pas. On filtre sur les post-conditions.

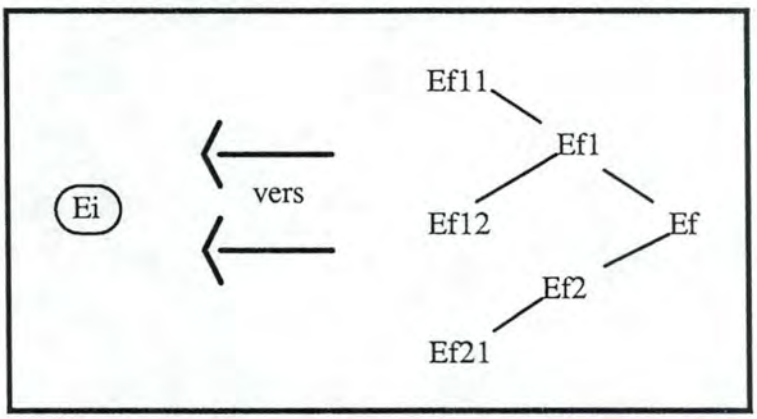


Figure 5.3 : Chaînage arrière.

Chaînage mixte

Chaînage mixte : On fait un peu de chaînage avant et de chaînage arrière; le problème est alors de savoir où sont les états initial et final.

Lorsqu'on planifie, on développe un arbre de recherche dont les noeuds sont des états, et les branches des opérations; le plan est, par conséquent, la suite de branches qui relie Ei à Ef. Les trois points qui suivent sont trois façons de parcourir l'arbre de recherche.

Recherche en largeur d'abord

Recherche en largeur d'abord : on parcourt toutes les racines de l'arbre qui sont sur le même niveau avant de descendre de niveau (Figure 5.4). On est sûr de parcourir toutes les racines, donc d'aboutir à une solution (un plan ou échec). Cette technique, quoique très sûre, n'en est pas moins fort pesante au niveau du temps de recherche.

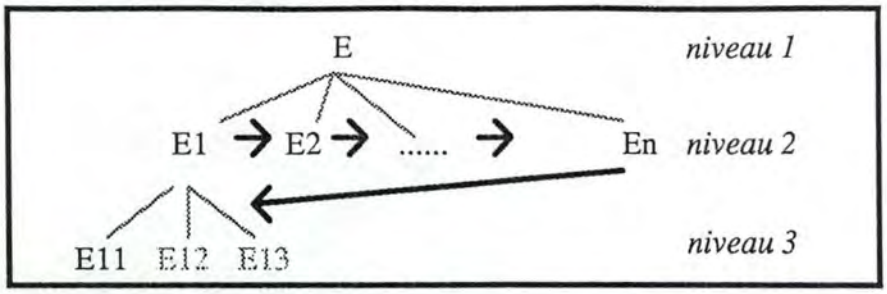


Figure 5.4 : Recherche en largeur d'abord.

Recherche en profondeur d'abord

Recherche en profondeur d'abord : on descend dans l'arbre de recherche par un noeud le plus éloigné possible. Si on est bloqué, on remonte au noeud frère du niveau supérieur (figure 5.5).

Ce type de planification est très dangereux, on risque fort de s'enfermer dans une boucle et de ne jamais pouvoir en sortir. Elle demande donc des mécanismes de surveillance de l'évolution du graphe qui permettent de relancer la recherche si on voit qu'elle s'enlise.

Les opérations de création d'objets sont, pour la plupart, sans pré-conditions; on peut donc les appliquer dans tous les cas. A ce titre, elles sont un ennemi important de cette méthode de recherche.

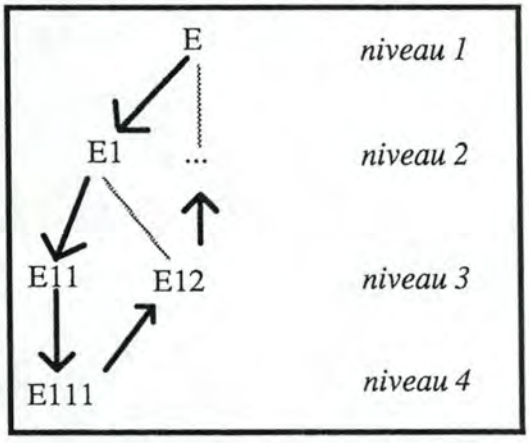


Figure 5.5 : Recherche en profondeur d'abord.

Recherche ordonnée (heuristique)

Recherche ordonnée : on parcourt la racine qui a le plus de chance d'aboutir à un plan.

Pour ce faire, on développe à côté du planificateur un système de scoring des plans d'après des heuristiques (par exemple : les plans les plus courts sont privilégiés, les plans qui sont le plus près de la solution (état modifié fort proche de l'état final ou initial suivant chaînage) sont privilégiés, ...). Si cette racine conduit à une impasse, on remonte au niveau supérieur et on continue avec la meilleure racine non encore visitée.

C'est cette technique associée au chaînage mixte qui sera utilisée dans notre planificateur.

5.2. La planification dans Multiworks

5.2.1. Rôles

En rentrant dans le planificateur, on dispose d'un état initial valide, d'un liste d'opérations sur la tâche, et d'un ensemble plus ou moins étoffé d'état finaux potentiels[37]. Pour simplifier le travail, on considérera que la planification s'attache à joindre l'état initial avec un seul état final. Pour lever cette réduction, il suffit d'appliquer le planificateur à chaque état final, et ensuite choisir le meilleur plan, validant ainsi l'état final qui en découle[38].

La planification en plus de son rôle de recherche de séquence d'actions, se voit ajouter un rôle de validation et, le cas échéant, de choix d'état intentionnel espéré de l'utilisateur. Elle permet également de dégager un diagnostic devant faciliter l'explication de l'échec s'il y a lieu.

5.2.2. Objet de la requête

Au chapitre 2, nous avons repris la définition de la requête proposée par [Allen 80].

Request (speaker, hearer, action)

; le speaker demande à l'hearer de faire l'action.

Nous avons dit ensuite, qu'une requête avait comme conséquence de placer l'objet de la requête dans l'état intentionnel de l'interface et de lancer la planification avec cet état intentionnel comme état final.

L'objet de la requête, pour [Allen 80], [Litman 87] est une action. Est-ce à dire que la finalité de la requête soit l'exécution de l'action indépendamment du résultat de cette action ou de critères temporels ?[39] Nous ne le pensons pas. Une action existe pour faire évoluer l'état et ne se satisfait pas à elle-même.

Pourquoi dès lors reprendre l'action comme objet de la requête et non pas un état escompté ? La réponse à cette dernière question dépend à notre avis de l'acteur de la planification et donc de la responsabilité de celui-ci.

En guise d'illustration, revenons un instant à la pâtisserie. Quand on passe commande d'un gâteau au chocolat, on ne demande jamais à la vendeuse :

---

[37] J'ai déjà précisé que les modules de traitement du langage, avaient comme objectifs de se saisir des hypothèses de sortie des niveaux inférieurs, de les filtrer et de ressortir avec des hypothèses de niveau supérieur. On considère que le module précédant la planification a comme finalité ultime de traduire les actes de langage potentiellement repris par l'énoncé en un certain nombre d'états finaux espérés.

[38] Ce choix n'est certainement pas trivial. Comment le formaliser ? Sur quels critères? En choisissant un plan, et donc un état final, on pose clairement que, pour nous, c'est non seulement le meilleur mais aussi le seul valide. Un fois appliqué, on pourra difficilement revenir sur ses pas .

[39] Ainsi la requête Request (X, Y, fermer_la porte) précéderait une fermeture de la porte qui, à la limite, peut ne jamais se terminer et qui n'accorde aucune importance au fait de savoir si la porte sera fermée un jour; plutôt que le passage de l'état ouvert à l'état fermé le plus vite possible.

"Prends des oeufs, du sucre, de la farine, ..."

" Mélange la farine avec les oeufs, ..."

"Mets tous cela au four pendant 30 min."

"..."

On se contente de dire : "je voudrais un gâteau au chocolat". Ce n'est pas à nous à planifier la création du gâteau, la requête se centre dès lors sur le résultat. La composante principale de l'énoncé est dans ce cas, clairement l'objet : "un gâteau au chocolat".

Par analogie, la plupart des utilisateurs débutants de système informatique agiront de la même façon par leurs requêtes.

Comment se fait-il alors que pour beaucoup de systèmes actuellement opérationnels ou en développement, la partie centrale de l'énoncé soit le prédicat, autrement dit le verbe(cfr. le projet DIAPASON [Souvay 92] et aussi [Klein 90]).

Ces systèmes sont développés autour de grammaires de cas (définition de cas, d'arrangements de compléments possibles autour d'un verbe), et sont, avant tout, des systèmes demandant une grande maîtrise technique (DIAPASON est un système de commande de sonar) incompatible avec une planification libérale de la machine ou des systèmes où ce qui importe n'est pas tellement le résultat mais plutôt la façon dont on arrive à celui-ci. Dans ce dernier cas, on voit mal comment on pourrait faire découvrir le plan que l'on voulait voir appliqué par la machine, en ne se basant que sur les résultats[40].

Le projet Multiworks est complexe car il recouvre ces deux points de vue. D'une part, il doit pouvoir répondre (en ouvrant groupe, noeuds, ...; positionnant les fenêtres de façon optimale, ...) à des requêtes du genre :"donne-moi les informations sur le Canari des Iles". D'autre part, il doit correctement exécuter des requêtes précises comme "Déplace la fenêtre en haut et à gauche", qui se rapproche plus de l'utilisation experte du logiciel.

Dans les deux cas, la planification jouera le même rôle; on devra ajuster la représentation des états finaux en se centrant plus spécifiquement sur le résultat ou sur l'action voulue. On voit ici tout l'avantage qu'il peut y avoir à utiliser le même formalisme pour représenter objets et actions. L'état intentionnel sera un état final composé d'objets et/ou d'actions instanciées.

Comment savoir à travers un énoncé que l'objectif de l'utilisateur est un résultat ou la façon d'y arriver? Le problème reste ouvert.

5.2.3. Les contextes dans notre modèle

Jusqu'à présent, les zones temporelles sont fondamentalement de deux types : réelles ou hypothétiques (Cfr. chapitre 4).

---

[40] Dans une application de pilotage d'un robot, on voit mal comment la machine pourra retracer le parcours de celui-ci en recevant seulement ses situations de départ et d'arrivée.

La planification nécessite, à côté de ces zones réelles, d'établir une certaine gradation (temporelle) dans l'hypothétique. L'introduction des contextes au modèle permet une telle gradation.

Un **contexte** est une liste ordonnée d'actions instanciées (actions + substitutions).

A chaque zone est associé un contexte. On le représente graphiquement par une liste qu'on positionne en dessous de la zone concernée (Cfr. Figure 5.6). C'est la liste d'actions (le plan) qu'il faudrait exécuter pour que la zone devienne réelle (Cfr. Figure 5.7).

Les zones réelles auront un contexte vide : ().

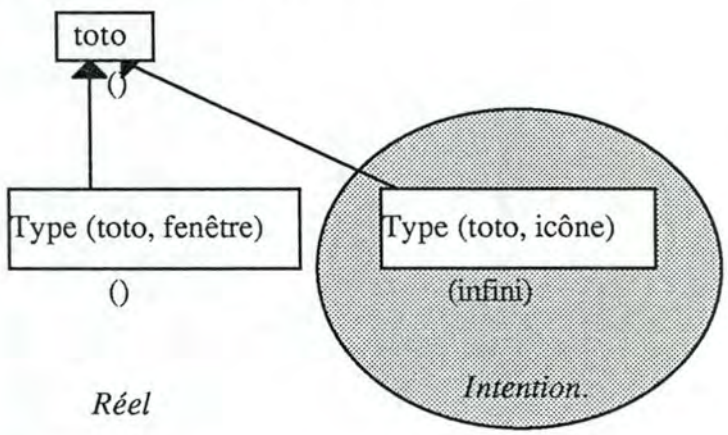


Figure 5.6

A priori, on ne connaît pas la séquence d'actions, le plan, donc le contexte associé aux zones représentant des intentions. C'est même pour cela qu'on développe un planificateur. On va donc leur associer un contexte indéterminé, que l'on notera (infini). Le choix de cette notation se justifie par le fait qu'on considère qu'a priori, il faudrait exécuter une liste infinie (car non déterminée) d'actions pour valider la zone intentionnelle

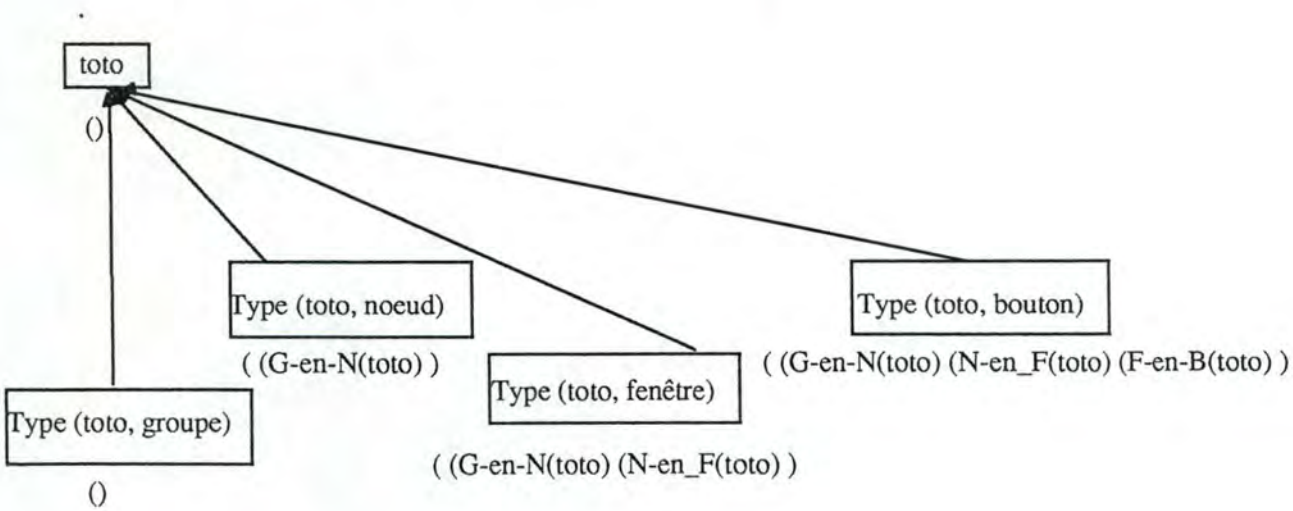


Figure 5.7

La figure 5.7 signifie que l'objet toto sera un bouton si son type passe de groupe à noeud, de noeud à fenêtre et de fenêtre à bouton.

En introduisant les contextes, on introduit une gradation dans le degré hypothétique des zones.

5.2.4. Passé, présent, futur dans le modèle

Le passé et le futur n'existent que par rapport au présent. Celui-ci est défini par l'ensemble des zones valides (les plus à droites des chaînes de précédences et de père valide) dont le contexte est vide. Le passé aura aussi son contexte vide et sera composé de l'ensemble des zones précédant le présent. Le futur sera composé des zones à contexte non vide (Cfr. Figure 5.8).

Tant dans le passé (position dans la chaîne d'adjacence) que dans le futur (nombre d'éléments du contexte), on retrouve une temporalité. Les zones les plus lointaines du futur sont celles dont le contexte se limite à (infini).

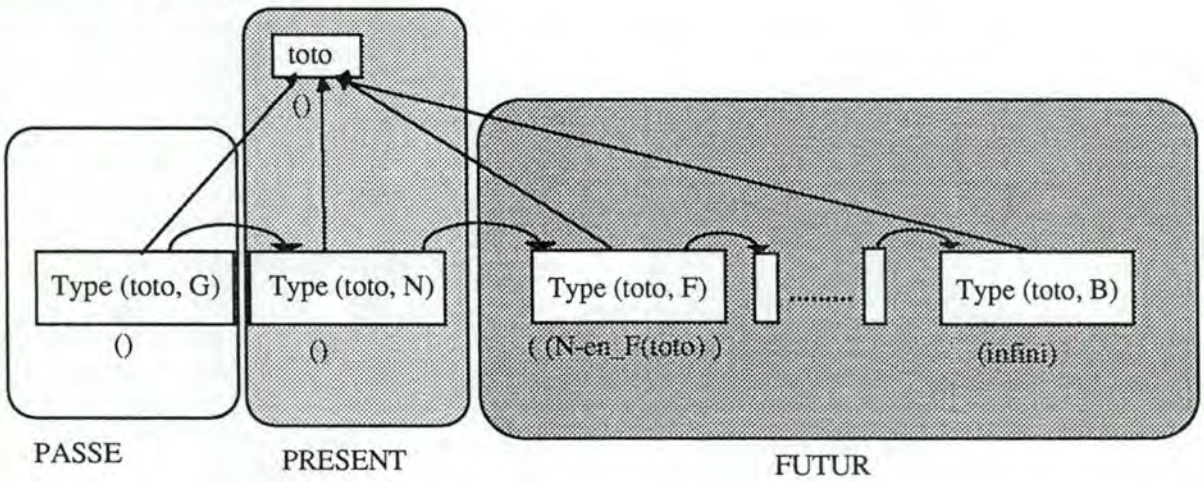


Figure 5.8

L'objectif de la planification sera de faire le lien entre le présent (temps de lancement de l'algorithme) et le futur "infini".

5.2.5. Planificateur Multiworks

Objectifs et moyens

Le planificateur travaillera sur les zones temporelles et utilisera les concepts définis précédemment (chapitre 4) et les fonctions déjà implémentées pour la recherche des référents (avec modifications éventuelles).

Nous décrirons les algorithmes du planificateur dans une logique fonctionnelle d'inspiration LISP.

Le planificateur doit permettre le diagnostic de l'échec (chapitre 6).

L'état de la tâche est une variable globale du système, la planification agissant directement sur cette représentation des objets, on doit donc toujours pouvoir y revenir. L'utilisation des contextes associées au zones temporelles le permet.

Nous utiliserons une méthode de recherche ordonnée dans l'arbre de planification, en chaînage mixte, pour les raisons dont nous nous sommes expliqués.

Concepts et mécanismes

Les objets et les actions sont modélisés par des zones temporelles . A chaque zone des objets, on associe un contexte (chapitre 4).

On rangera les objets dans une liste. Cette liste sera composée des zones temporelles contenant leurs identifiants.

Au début de la planification, l'état initial est composé de l'ensemble des zones temporelles (ZT) valides[41] de l'état qui ont un contexte vide : (). L'objectif sera constitué de l'ensemble des ZT à contexte = (infini).

Utilisant la technique du chaînage mixte, un plan sera de la forme (**pav, par**) tel que **pav** est un plan obtenu en chaînage avant et **par**, en chaînage arrière. **Par, pav** sont des contextes. Au départ, **pav** = () et **par** = (infini). On complète **pav** par la droite et **par** par la gauche[42].

Une **zone réelle** est une zone qui ne contient pas "infini" dans son contexte. On l'oppose à **zone objectif**.

Au fur et à mesure qu'on avance dans la planification, on complète le contexte des nouvelles zones réelles par la droite et des nouvelles zones objectifs par la gauche (Figure 5.9).

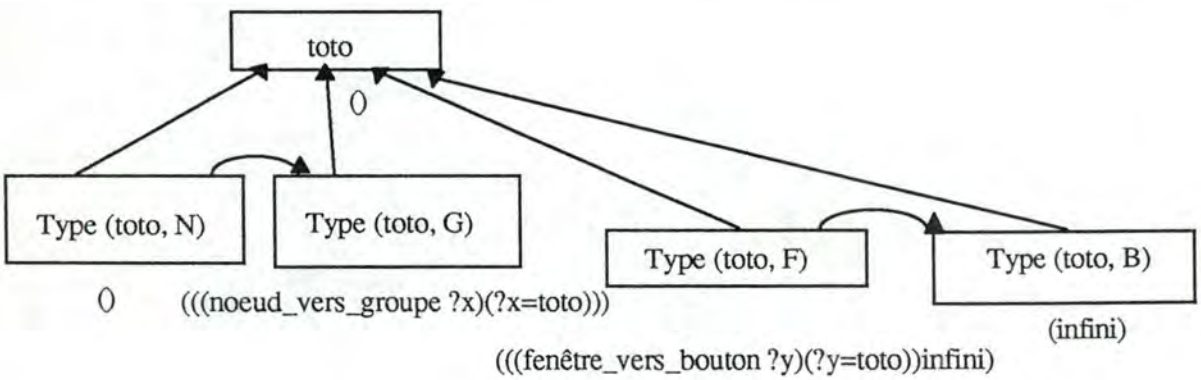


Figure 5.9

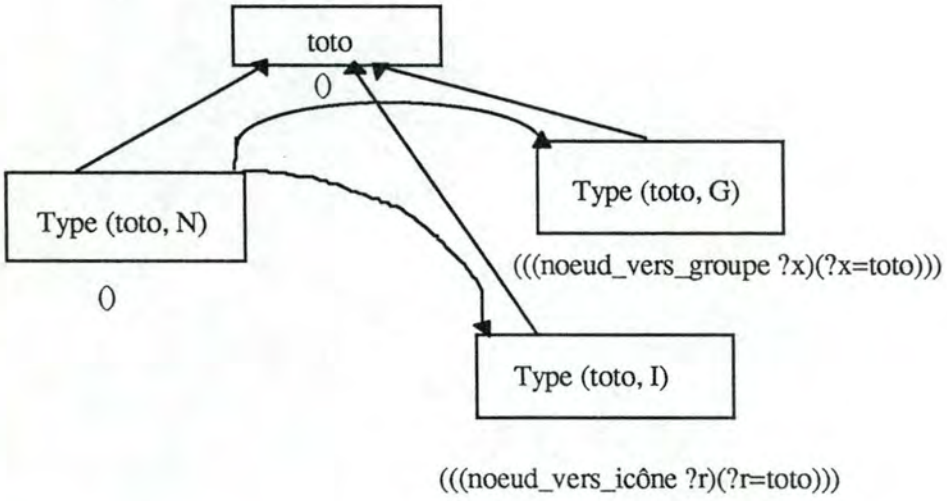
[41] Nous avons défini ce qu'on entendait par zones valide au chapitre 4.

[42] Il ne faudra pas oublier d'enlever "infini" de **par** pour retrouver le plen final.

Le problème majeur en chaînage mixte est de retrouver Ei et Ef (objectif, pour nous) en cours de planification. Pour un (**par, pav**) donné, l'état initial est l'ensemble des zones valides les plus à droites possibles comprises dans le contexte **pav** (préfixes de **pav**); l'objectif est l'ensemble des zones objectifs les plus à gauche possibles comprises dans le contexte **par** (suffixes de **par**).

Une zone est comprise dans le contexte **pav** si son contexte est un préfixe de **pav**.

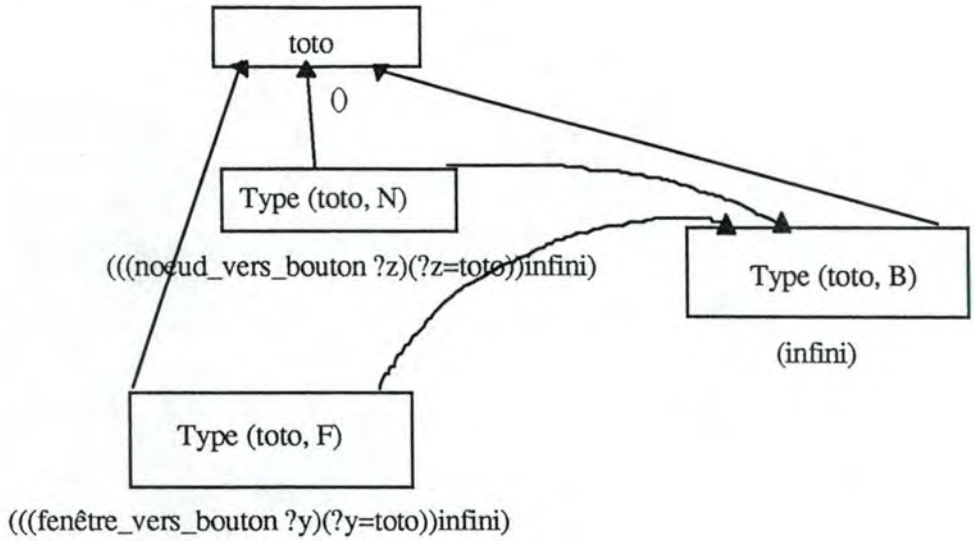
Ex:



Si **pav** = (((noeud_vers_icone ?r)(?r toto))), les zones comprises dans le contexte sont toto, Type (toto, N) et Type (toto, I). Cependant, on ne recherchera d'équivalence à l'objectif qu'avec les zones les plus à droites : toto et Type (toto, I).

Une zone objectif est comprise dans le contexte **par** si son contexte est un suffixe de **par**.

Ex:



Si on a **par** = (((noeud_vers_bouton ?z)(?z toto)) infini), les zones Type (toto, B) et Type (toto, N) sont comprises dans le contexte.

La zone objectif est Type (toto, N) car c'est la zone la plus à gauche du contexte.

Un **score** est une valeur associée à un chemin de l'arbre de planification. Plus cette valeur est élevée et plus ce chemin a de chance d'aboutir à un plan qui réponde aux objectifs.

En général, on construit l'arbre de planification (chaînage mixte) en partant d'un état initial et d'un objectif de niveau A, en appliquant les actions et avant et en arrière à ce niveau, pour ressortir avec un ensemble de nouveaux états initiaux ou objectifs au niveau inférieur (B). Le plan est alors la concaténation du plan de niveau A avec l'opération choisie pour descendre de niveau (ajoutée à **pav** ou **par** selon le chaînage choisi) (figure 5.10).

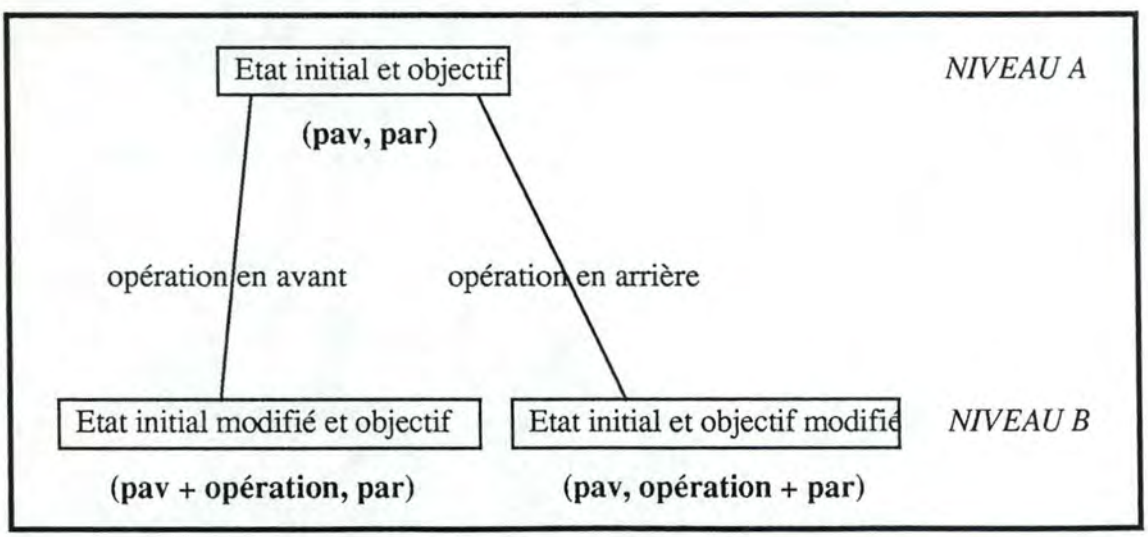


Figure 5.10 : Arbre de recherche.

L'exécution des opérations a comme effet d'ajouter des zones à l'état. Ces zones étant pour la plupart inutiles (branches non visitées), et ne disposant pas de mécanisme pour les enlever, on affine alors la méthode de construction de l'arbre tout en assurant notre objectif de "recherche ordonnée".

Au lieu d'appliquer les opérations, on préférera anticiper leur comportement sur l'état initial et l'objectif : regarder si l'opération est applicable, le nombre de zones qu'elle ajouterait, le nombre d'objectifs qu'elle atteint, sa nature (on se méfiera des opérations de création d'objets qui peuvent conduire à créer des "choses" non désirées), ... On utilisera nos conclusions et le score du noeud père pour établir le score des noeuds du niveau B. Le plan, indépendamment de toute exécution d'opération, pourra en être inféré.

On triera ensuite les noeuds de niveau B, et on n'appliquera que l'opération du plus probable, déterminant ainsi un nouvel état initial ou objectif père.

Un univers de la planification (**univers**) se compose d'une paire de contexte (**pav par**) et d'un **score.**

Grâce à l'état (qui est une variable globale du système) et à ces deux contextes, on sait retrouver l'état initial et l'objectif à ce stade dans la planification .

A chaque noeud de l'**arbre** de planification, on associe un **diagnostic** ou un **univers.**

Un **diagnostic** est une liste composée d'un **univers** (univers père de ce noeud), d'une opération instanciée (c'est à dire opération et substitutions sur les objets de l'état), du **score** de ce noeud dans l'arbre et d'une liste de commentaires sur la simulation de l'exécution de cette opération sur les zones des paramètres comprises dans les contextes de l'**univers** (pré-conditions vérifiées, non vérifiées; post-conditions vérifiées, non vérifiées; ...).

Nous définissons l'**arbre** de planification comme suit : il s'agit d'une liste de noeuds dont le premier est l'**univers** de la planification et les autres des **diagnostics**. La liste des **diagnostics** peut être vide.

Plan

Présentons les différentes fonctions internes au planificateur. Nous travaillerons d'après une méthode Top-Down. Pour chacune d'elle, et dans la mesure du possible, nous décrirons son fonctionnement, donnerons un algorithme[43] d'inspiration LISP.

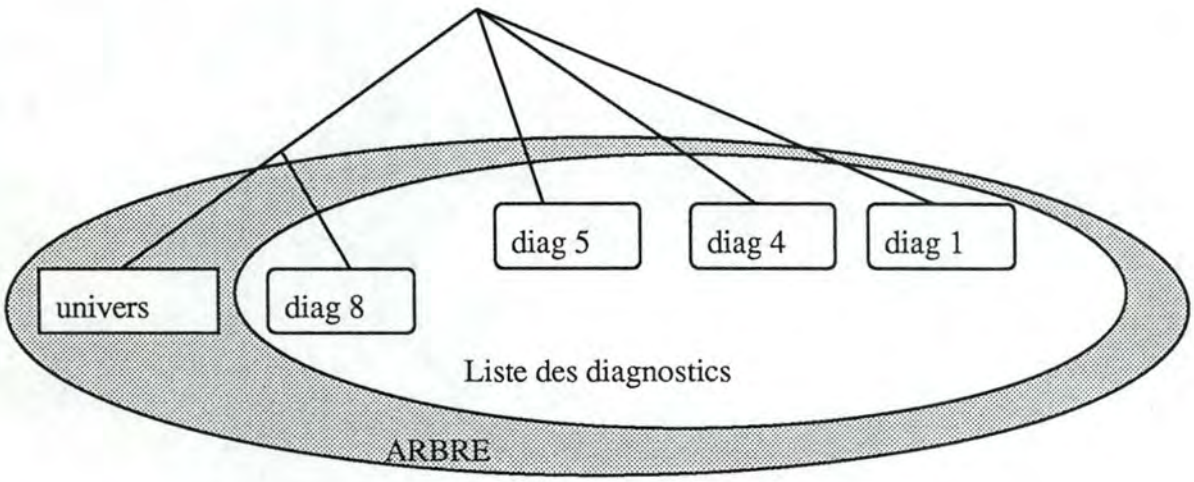
---

[43] Remarques :

- Les fonctions commencent par une majuscule, les variables par une minuscule.
- (Fonction v1 ... vn) : retourne le résultat de l'application de Fonction avec comme arguments : v1...vn.
- Les opérateurs logiques (si, alors, sinon, pour chaque...de...) seront soulignés.
- Variable = valeur : signifie que la variable prend la valeur.

Fonctionnement [44]

On a, en entrant dans le planificateur, un arbre de la forme :



On rentre dans le planificateur avec l'**arbre** qui contient au minimum l'**univers** (la **liste des diagnostics** peut être vide).

Dans les **diagnostics**, on retrouve l'**univers père**, l'**opération** qui permet de descendre du père au fils, des **commentaires** et un **score**.

On a trié la **liste des diagnostics** dans l'ordre décroissant des **scores** .

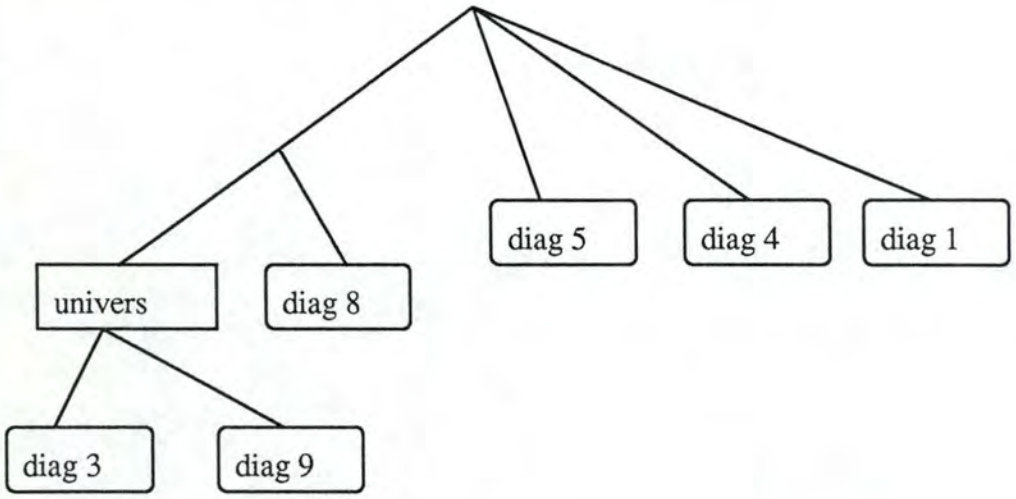
On vérifie d'abord si on n'a pas trouvé la solution (tous les objectifs sont atteints dans **l'univers**).

Si oui, on arrête et le plan est la concaténation des contextes de **l'univers** (ne pas oublier d'enlever "infini" !).

Sinon, on distribue les actions sur l'état compris dans les contextes de **l'univers**, on construit les **diagnostics** et on calcule les **scores** qui découlent de cet univers (d'où la nécessité de conserver le score de la branche dans ce dernier) (on obtient diag 9 et diag 3) .

---

[44] Remarque : l'univers est modélisé par un rectangle, les diagnostics par des rectangles à angles arrondis (on y indique le score).



On peut imaginer que le **score** soit mis à (- infini) si l'action n'est pas applicable ou que ce **diagnostic** est oublié. On préférera la deuxième solution pour des gains évidents de place de stockage. Néanmoins, dans le cas où la planification ne donne rien, il est nécessaire de conserver les **diagnostics** qui ne vont pas pour pouvoir expliquer l'échec de la planification.

De toute façon, il faudra faire le tri entre les **diagnostics** pertinents pour expliquer l'échec et les autres.

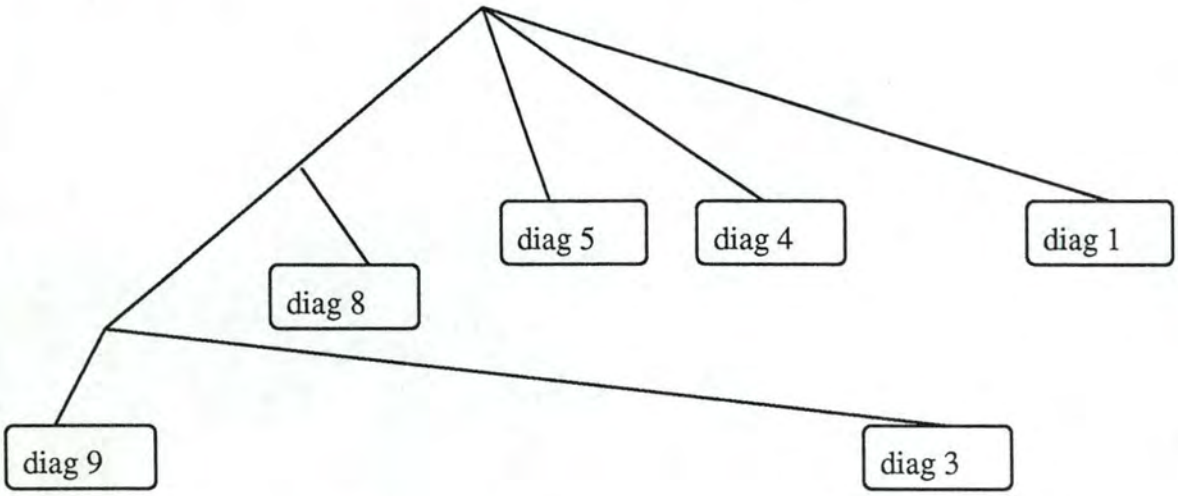
On peut donc se limiter à ce que :

1) Un module spécial (externe au planificateur) ne retienne que ceux qui pourraient expliquer l'échec.

2) On ne reprenne plus les **diagnostics** qui échouent dans la liste des diagnostics pour la planification proprement dite.

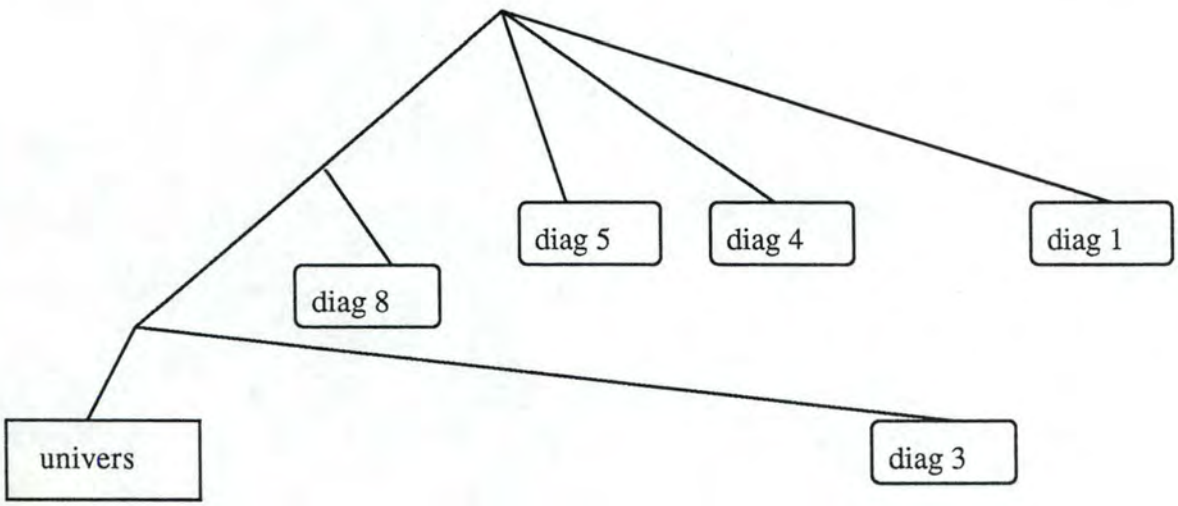
On ajoute les nouveaux diagnostics (diag 9, diag 3) à la liste déjà existante (diag 8, diag 5, diag 4, diag 1)

On trie ensuite la liste des **diagnostics** dans l'ordre décroissant des **scores**.



Si à ce stade, l'opération reprise par le **diagnostic de poids le plus élevé** est inapplicable, on peut arrêter la planification, on ne trouvera pas de solution (on a pris soin de mettre les **diagnostics** qui "fonctionnent" en tête de liste).

Sinon, on applique l'opération à **l'univers père** du **diagnostic premier de liste** (d'où la nécessité de le conserver dans chaque diagnostic) et on détermine ainsi un **nouvel univers**.



On rentre alors dans le planificateur avec le nouvel arbre.

Algorithme

Plan (tree)[45]

si (<> 'échec (Trouvé_plan (Univers_arbre tree)))

alors (Trouvé_plan (Univers_arbre tree))

sinon diag_univers = (Diagnostiquer (Univers_arbre tree))

liste_diag = (Ordonner_diagnostics (Concaténation diagnostics_univers (Diagnostics_arbre tree)))

si (Continuer_plan liste_diag)

alors univers_modifie=(Modifier_etats (Premier_diag liste_diag))

niew_tree=(Concaténation univers_modifie (Reste_diag liste_diag))

(Plan new_tree)

sinon (Echec_plan liste_diag)

Avec:

NOM	TYPE
tree	arbre
new tree	arbre
Univers_arbre	univers
univers_modifié	univers
Modifier_état	univers
Ordonner_diag	liste de diagnostics
liste_diag	liste de diagnostics
Diag_univers	liste de diagnostics
Reste_diag	liste de diagnostics
Diagnostics_arbre	liste de diagnostics
Premier_diag	diagnostic
Trouvé_plan	contexte

[45] La première ligne en gras de l'algorithme reprend le nom et les paramètres de la fonction.

Modules triviaux

Univers_arbre, Diagnostics_arbre, Diag_univers, Premier_diag, Reste_diag sont des fonctions d'accès à des champs de données.

Concaténation est une fonction qui, comme son nom l'indique, donne la concaténation entre deux éléments.

(<> a b) retourne la valeur vraie ssi a est différent de b.

Trouvé_plan

Fonctionnement

A partir des deux contextes **pav** et **par**, cette fonction retourne la valeur *'échec'* s'il existe au moins une zone objectif la plus à gauche possible et dans le contexte **par** qui ne trouve pas d'équivalent dans les zones réelles les plus à droite possibles et comprises dans le contexte **pav**. Sinon elle retourne le plan.

Algorithme

Trouvé_plan (pav par)

res = ()

pour chaque objet de état :

liste_objectifs = (Rechercher_objectifs objet par)

pour chaque objectif de liste_objectifs:

res = (Concaténation (Recherche_nouveauté_avec_contexte objet objectif pav) res)

si (Vide res)

alors (Concaténation pav (Tout_sauf_infini par))

sinon 'échec

Rem : On peut faire le test (Vide res) après chaque traitement d'un objectif et s'arrêter dès que c'est faux pour gagner du temps.

Modules triviaux

- Tout_sauf_infini retourne *par* sans le dernier élément.

- La fonction Recherche_nouveauté_avec_contexte est déjà existante.

- Rechercher_objectifs est définie en appelant la fonction Recherche_extrémités_contexte (objet pav backward). Nous spécifierons ces fonctions déjà implémentées plus tard.

Diagnostiquer

Opérations de modifications sur des objets existants

Ces actions sont des opérations avec paramètres et qui s'appliquent sur des objets qui ont leur contexte vide.

Fonctionnement

La fonction Diagnostiquer a comme but de produire la liste des diagnostics des applications des actions sur les objets de l'état compris dans les contexte de l'**univers.**

Pour chaque action, on recherche tout d'abord l'ensemble des objets pris comme paramètres. Cette recherche de l'ensemble des combinaisons possibles des objets pouvant subir l'action est facilitée par plusieurs éléments.

On peut tout d'abord considérer que chaque objet sera d'un type donné et que la description des actions reprendra la liste des types de ses paramètres (le type d'un objet pourra évoluer au cours du plan, on saura néanmoins toujours le retrouver si on dispose de l'univers!).

On peut, ensuite, imaginer de représenter l'état par un ensemble de sous-états contenant des objets d'un type donné.

On peut enfin espérer, mais ce n'est toutefois pas toujours le cas que, pour aboutir à un résultat, les actions ne modifieront que les objets qui contiennent au moins une zone objectif; on ne reprendra donc pas les objets statiques dans l'état et les actions qui s'appliquent uniquement à ces derniers dans la liste d'actions.

Algorithme

Diagnostiquer(univers)

liste_diag = ()

pour chaque action de liste_actions

  liste_parametres = (Sélectionner_paramètres action univers)

  pour chaque paramètre de liste_paramètres

diag_act = (Diagnostiquer_action action paramètre univers)

si (Conditions_diagnostic diag_act)

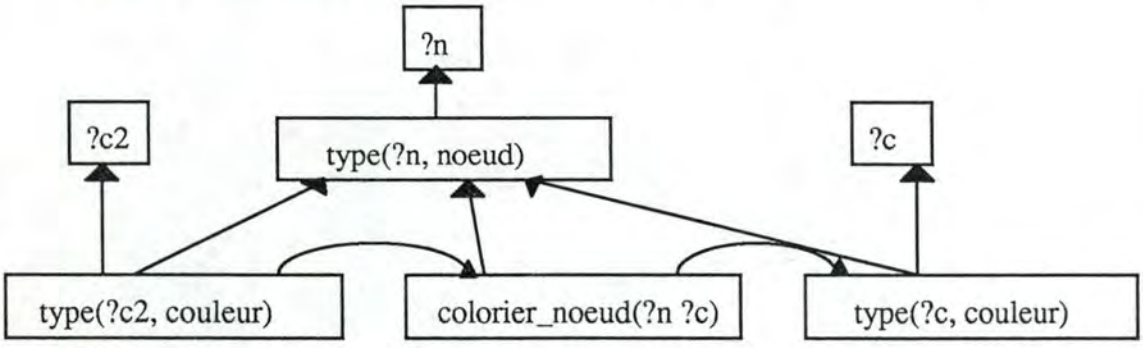
alors liste_diag = (Concaténation diag_act liste_diag)

Sélectionner_paramètres

Fonctionnement

Cette fonction permet de faire un premier tri dans les diagnostics (on ne recherche pas à appliquer une action à n'importe quel objet).

Ex : soit l'opération de mise en couleur d'un noeud : (colorie_noeud ?n ?c).



Dans la description de cette action, on voit que (type ?n noeud) et (type ?c couleur); où le prédicat (type nom_objet nom_type) signifie que l'objet nom_objet est de type nom_type.

Si on dispose d'une clé d'accès au sous-état de type donné telle que (Type-nom_de_type état) = liste de nom_objet et d'une fonction de distribution des éléments de ces sous-états entre eux, on obtient facilement la liste des paramètres possibles de l'action.

Il n'est pas possible de mettre physiquement les objets dans des sous-listes de type donné car un objet peut voir son type évoluer au cours de la planification (un noeud peut devenir une icone, ...) et l'état doit rester une variable globale.

La solution qui se propose alors à nous est d'appeler une fonction qui reçoit un nom_type, un univers et qui parcourt l'état pour donner cette sous-liste au moment souhaité.

Algorithme

Sélectionner_paramètres (action univers)

liste_paramètres_action = ()

pour chaque action_type_paramètre de action

liste_objet = (Trouve_objets action_type_paramètre univers)

liste_paramètres_action = (Distribue liste_objets liste_paramères_action)

liste_paramètres_action

Trouve_objets

Algoritme

Trouve_objets (nom_type univers)

liste_objets = ()

pour chaque objet de état

extrémités = (Recherche_extrémités_contexte objet (Pav_univers univers))

si (Member "type nom_type objet" extrémités)

alors (Concaténation objet liste_objets)

liste_objets

Remarques :

- Recherche_extrémités_contexte est une fonction déjà existante qui donne l'ensemble des zones incluses dans une zone de départ les plus à droites (à gauche si argument supplémentaire : backward) possibles et dans le contexte.
- Pav_univers est une fonction d'accès au champ pav de univers.
- Member est une fonction qui dit si un élément (spécifié explicitement entre ") fait partie d'une liste.

Distribue

Fonctionnement

Cette fonction reçoit deux arguments : une liste d'éléments et une liste de liste d'éléments .

Soient donc LE=(e_1 .... e_n) avec n>=0

et LLE = (le_1 .... le_m) avec m>=0 ou LLE='echec.

Si n=0 ou LLE='echec

alors RES='echec

sinon RES=(e_1+le_1 ... e_1+le_m...e_i+le_1...e_i+le_m...e_n+le_1...e_n+le_m)

Remarque : e_i+le_j signifie qu'on ajoute l'élément e_i à la liste le_j.

Diagnostiquer_action

Fonctionnement

A partir d'une opération, des paramètres et d'un univers, cette fonction prédit de la vérification des pré-conditions, de la complétion des objectifs des paramètres.

En fonction des résultats de cette prédiction, on rejetera la branche de l'arbre (non-respect des pré-conditions et post-conditions non toutes atteintes), on modifiera le score (le score peut être le nombre d'objectifs atteinds), on indiquera qu'il faut appliquer cette opération en chaînage avant (pré-conditions toutes vérifiées indépendamment des post-conditions) ou en chaînage arrière (tous les objectifs des objets paramètres sont atteints et non respect de toutes les pré-conditions).

pré conditions	**post conditions**	**diagnostic**
vérifiées	non vérifiées	• chaînage avant • score=score + nb objectifs atteints • opération normale
vérifiées	vérifiées	• chaînage avant (c'est un choix !) • score=score + nb post conditions • opération normale • objectifs atteints pour paramètres

non vérifiées	vérifiées	• chaînage arrière • score =score + ? • opération inverse
non vérifiées	non vérifiées	échec

Algorithme

On utilisera la fonction déjà définie ActionEstApplicable; ensuite, en fonction des résultats de cette dernière, on établira le diagnostic comme indiqué par le tableau.

Conditions_diagnostic

Vérifie que diag_action <> 'échec.

Opérations de suppression sur les objets existants

(Il convient de définir d'abord ce qu'on entend par suppression; si elle consiste à ajouter un prédicat "inactif" à l'objet, cette opération peut être traitée comme une modification).

Opérations de création d'objets

Ces opérations sont d'un type particulier. Elle seront en effet toujours possible, ce qui peut conduire à gonfler l'état inutilement et occasionner une perte de temps considérable.

Ce type d'opération doit donc être traité différemment.

Ordonner_diagnostics

Fonctionnement

Cette fonction reçoit deux listes de diagnostics : celle provenant du parcours plus avant de l'univers (la nouvelle) et l'ancienne. L'ancienne liste est déjà triée; la nouvelle, non.

Il suffit donc de parcourir les deux listes simultanément et d'inclure les éléments de la nouvelle dans l'ancienne à la bonne place.

Algorithme

Ordonner_diagnostics (nouvelle ancienne)

si (vide nouvelle)

alors ancienne

sinon si (vide ancienne)

alors nouvelle

sinon si (Score_supérieur (Premier nouvelle) (Premier ancienne))

alors (Concaténation (Premier nouvelle) (Ordonner_diagnostics (Reste nouvelle) ancienne))

sinon (Concaténation (Premier ancienne) (Ordonner_diagnostics nouvelle (Reste ancienne)))

Remarques :

- Premier donne le premier élément d'une liste;
- Reste, le reste de la liste
- Score_supérieur prend la valeur vraie si le score du premier paramètre est > au score du deuxième.

Continuer_plan

Comme on a choisi de ne reprendre dans la liste des diagnostics que les diagnostics des opérations qui fonctionnent, il suffit de vérifier que la liste des diagnostics n'est pas vide.

Modifier_états

Fonctionnement

Regarder d'abord si on est en chaînage avant ou arrière. On adapte alors l'application de l'opération.

Les fonctions d'application d'une opération existent déjà pour le chaînage avant; il suffit donc de les appeler (en fait, il faut introdruire le contexte !).

Pour le chaînage arrière, on peut conserver les mêmes fonctions de création de zones mais cette fois, il faut créer les zones réelles (et pas les zones hypothétiques); on devra, en plus, définir une fonction qui permet de relier une zone à une autre avec une relation de précédence (et plus de succession).

Il ne faudra pas oublier de définir le contexte des nouvelles zones créées :

- en chaînage avant, il s'agit du contexte de la zone qui précède suivi du couple (opération substitution);
- en chaînage arrière, ce sera le contexte de la zone qui suit précédé du couple (opération substitution).

Algorithme

Modifier_etats (univers_père opération score commentaires)

si (Chaînage_avant commentaires)

alors (Appliquer_action_avec_contexte_avant univers_père opération)

univers = (Concaténation (Concaténation (Pav univers_père) opération) (Par univers_père)score)

sinon (Appliquer_action_avec_contexte_arriere univers_père opération)

univers = (Concaténation (Pav univers_pére) (Concaténation opération (Par univers_père)) score)

univers

Echec_plan

Fonctionnement

Dans notre algorithme, cette fonction aura pour but d'informer le système que la planification n'admet pas de solutions, toutes choses égales par ailleurs.

Néanmoins, dans un système de dialogue, il est important d'ajouter en plus du fait que cela ne marche pas, des commentaires, observations sur le pourquoi de l'échec.

Nous préciserons plus longuement l'importance de cette fonctionnalité au chapitre 6.

5.3. Conclusion

Une requête de l'utilisateur ajoute un certain nombre de Zones Temporelles objectifs (contexte infini) au modèle et lance la planification.

Celle-ci permet, si possible, de déterminer une séquence d'actions qui, si elle est exécutée par l'application, rendra ces zones réelles (contexte vide).

En cas d'échec, le planificateur permet d'établir un diagnostic de celui-ci, qui devra, le cas échéant servir de support à un sous-dialogue d'éclaircissement.

Chapitre 6

6. Vers la génération de réponses

Concepts : **diagnostic, retour d'information, sous-dialogue, multimédia (modal), rhetorical acts.**

Jusqu'ici, nous avons réduit le rôle de la planification à trouver une suite d'actions qui modifie l'état de la tâche vers un état escompté. Si on trouve un plan, on valide par la même occasion l'état final; sinon, on doit se donner les moyens d'expliquer l'échec. La première partie du chapitre 6 y sera consacrée.

Nous verrons ensuite quels problèmes surgissent lorsqu'on ajoute aux actions sur la tâche du planificateur, des actions sur l'utilisateur. Nous préciserons les notions de multimédia, multimodal et d'action sur l'utilisateur indépendante de tout média ou mode. Cette deuxième partie nous amènera à faire le lien avec les concepts de plan du discours, de gestionnaire de sous-dialogue, ... rencontrées dans [Luzzati 90], [Litman 87]. Nous préciserons alors en quoi nos choix et notre démarche s'en écartent.

6.1. Agir sur la tâche

Le monde réel de l'interface est composé des actions et de l'état de la tâche.

6.1.1. La planification réussit

Le planificateur détermine une séquence d'actions qui relie le réel et l'intentionnel.

Etat final unique

Nous avons dit (Chapitre 2) que la reconnaissance du LN par une machine était un processus complexe divisé en modules hiérarchisés dont le but est d'envoyer des hypothèses au niveau supérieur en fonction d'un traitement interne sur les hypothèses de niveau inférieur[46].

[46] Nous simplifions.

Notre travail se situe au sommet du processus de reconnaissance. Le planificateur, recevra donc, en principe, plusieurs état finaux. Limitons, dans un premier temps, ce nombre à un.

Si un plan est trouvé, on supposera que le processus de reconnaissance a correctement fonctionné[47]. Le but de l'application étant d'agir, on exécute le plan et on limite notre réponse à cette exécution.

Plusieurs états finaux

Lorsqu'après planification, plusieurs états finaux restent possibles, ne sachant pas exécuter plusieurs plans simultanément (concurrence entre plans), on est bien obligé de choisir le meilleur donc, l'état final le plus probable.

Le problème de ce choix reste ouvert.

6.1.2. La planification échoue

La planification échoue lorsqu'aucun état final ne permet de trouver un plan. L'échec vient alors de l'une des deux parties; l'interface n'a pas su correctement décoder le message et/ou l'utilisateur ne s'est pas exprimé assez clairement.

Pour pouvoir expliquer l'échec, il faut faire en sorte que l'interface dispose de toute l'information sur le processus de reconnaissance qui vient d'avorter. Cette information doit être trouvée dans les différents modules de reconnaissance du LN. Le planificateur en fait partie. Il doit être à même de donner des explications sur le fond du problème.

Diagnostiquer l'échec

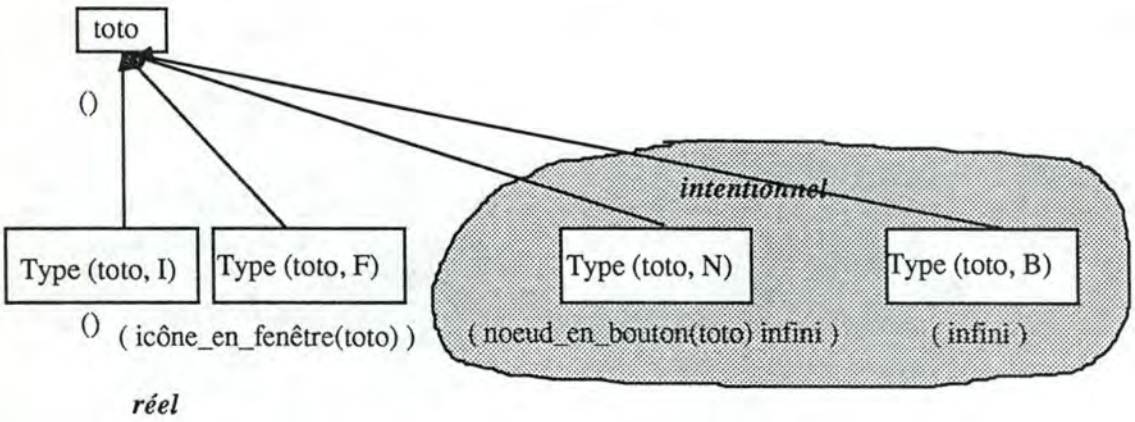
Le planificateur travaillant au niveau de la tâche, il ne saura donner d'informations que sur celle-ci, donc des informations sur le fond de la requête.

Il est capital qu'au cours de la planification, on conserve, en plus des plans potentiellement bons, ceux qui conduisent à une impasse. On a supposé que ces informations étaient rangées au fur et à mesure de la planification par un sous-module que nous n'avons pas spécifié.

Le diagnostic dans notre planificateur

Prenons un exemple :

[47] Le planificateur que nous avons développé s'arrête dès qu'il a trouvé un plan. A un état final donné correspondra donc au maximum un et un seul plan. On pourrait imaginer étendre le planificateur à tous les plans qui mènent à l'objectif, mais il faudra alors établir des critères de choix du meilleur plan.



L'utilisateur veut que l'objet toto (icône) deviennent un bouton. L'interface connaît deux actions : noeud_en_bouton et icône_en_fenêtre.

La planification va échouer bien évidemment. Grâce au chaînage mixte, les diagnostics reprendront les paires (**pav**, **par**).

L'échec est dû au fait qu'il n'existe aucune action qui relie ***pav*** à ***par***.

On peut expliquer l'échec de l'exemple précèdent en prenant le diagnostic qui contient (***pav***, ***par***) = (((icône_en_fenêtre(toto)), ((noeud_en_bouton(toto))); aucune action ne relie **pav** à **par** c-à-d qu'il n'existe pas d'action qui transforme une fenêtre en noeud.

A priori, aucun critère ne guide le choix du diagnostic; et certainement pas le nombre d'opération. La paire ((),(noeud_en bouton(toto))) laisse supposer qu'il n'existe pas d'opération qui transforme une icône en noeud, ce qui est tout aussi parlant.

Le nombre d'objectifs atteints est, à notre avis, un bon critère de choix du diagnostic. Ce problème reste, lui aussi, ouvert.

6.2. Agir sur l'utilisateur

Jusqu'à présent, nous avons considéré que l'interface agissait sur la tâche uniquement.

Les réponses aux requêtes de l'utilisateur nécessitent parfois des actions directes sur celui-ci.

Il en va de même lorsque, face à un échec, l'interface ne peut agir sur la tâche et se voit donc obligée d'agir sur l'utilisateur.

Le cloisonnement utilisateur-interface-tâche oblige l'interface à agir en parallèle sur la tâche et l'utilisateur pour garantir à ce dernier une actualisation de sa représentation de la réalité.

6.2.1. Utilisateur-interface-tâche

Le chapitre 3 était consacré à placer l'interface entre l'utilisateur et la tâche. Nous avons précisé que, du point de vue théorique, l'interface sert de sas entre les deux et que, par conséquent, il n'y a jamais de contact direct entre utilisateur et tâche.

Lorsque l'interface agit sur la tâche, pour que l'utilisateur se rende compte des changements et puisse, par conséquent, modifier son monde réel, il faut qu'elle les lui présente. Elle doit donc agir (on reste dans notre logique d'actions) sur l'utilisateur .

En ce qui nous concerne, MULTICARD (la tâche) entre en relation directe avec l'utilisateur par le gestionnaire d'écran et de fenêtre intégré. On mettra donc ce point de côté.

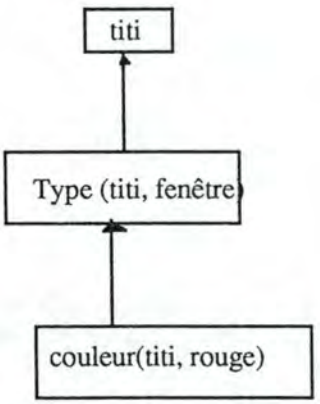
6.2.2. Répondre à des questions

Une question est par définition (Cfr. chapitre 2) un **Request (speaker, hearer, Inform)**.

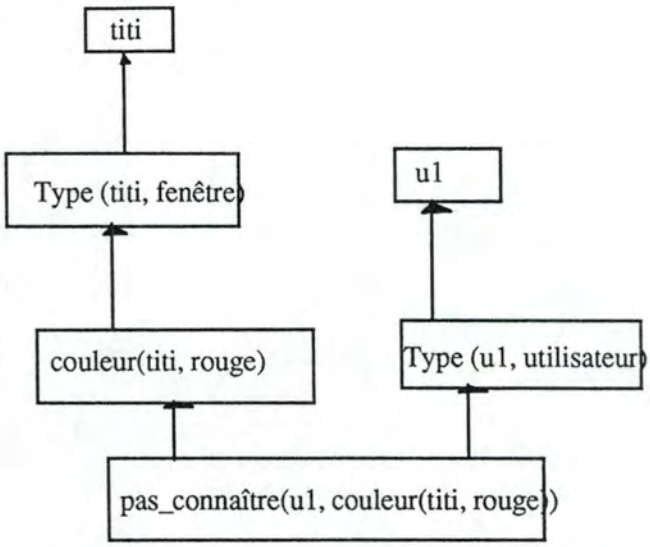
Inform est une action (speech act) non pas sur la tâche, mais sur l'utilisateur. Pour modéliser ce type de requête, il est indispensable d'ajouter des zones temporelles reprenant des informations sur l'utilisateur (modèle de l'utilisateur) au modèle de la tâche. De cette façon, on étend le monde réel de l'interface .

Exemple 1

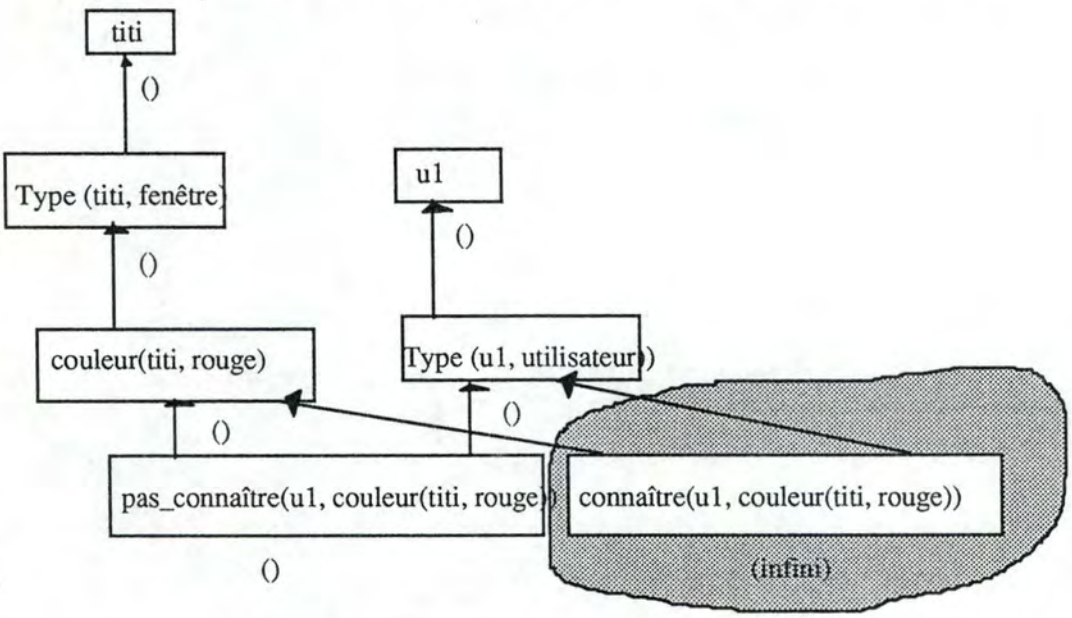
Soit titi, une fenêtre rouge :



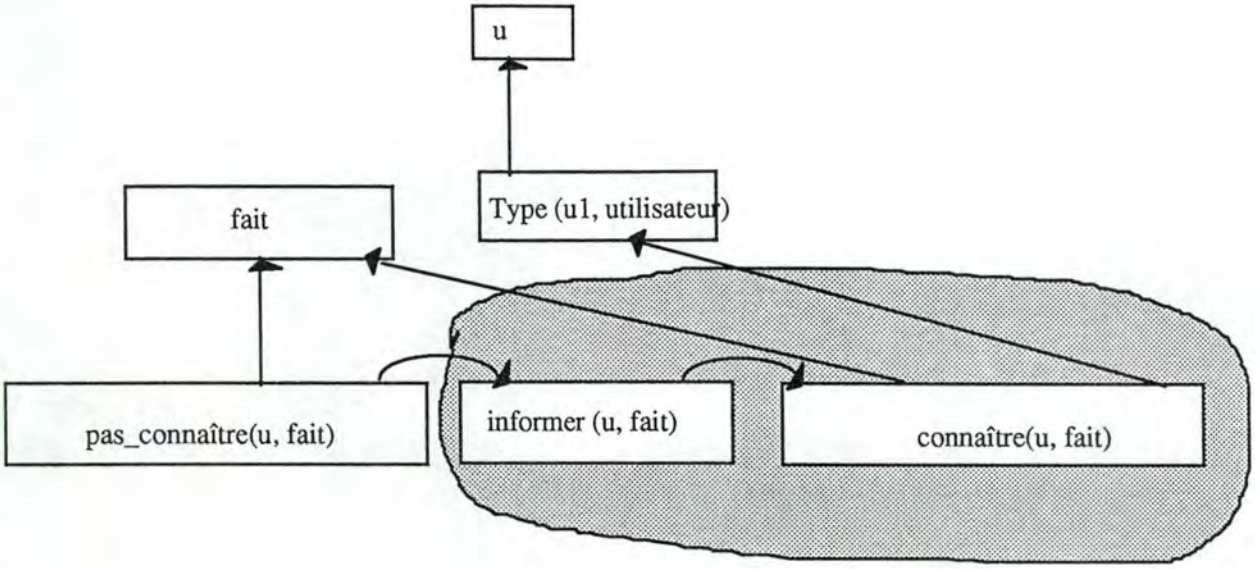
Soit l'énoncé : "*Quelle est la couleur de la fenêtre titi ?*" Cet énoncé comporte à notre avis deux actes. Le premier informe l'interface que l'utilisateur (identifié par u1) ne connaît pas la couleur de titi. Ce **Inform** a comme conséquence d'ajouter une zone réelle au modèle :



Le second acte de l'énoncé est une requête. Elle modifie le modèle comme suit :



Si on dispose de l'action informer(u, fait) :



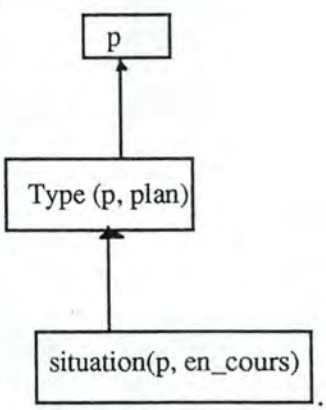
la planification ne posera aucun problème; le plan se composant de cette seule action.

6.2.3. Présenter l'échec

Admettons que la planification échoue et que grâce aux diagnostics, on trouve l'explication suivante : "*L'interface hésite entre deux fenêtres pour ranger un bouton*".

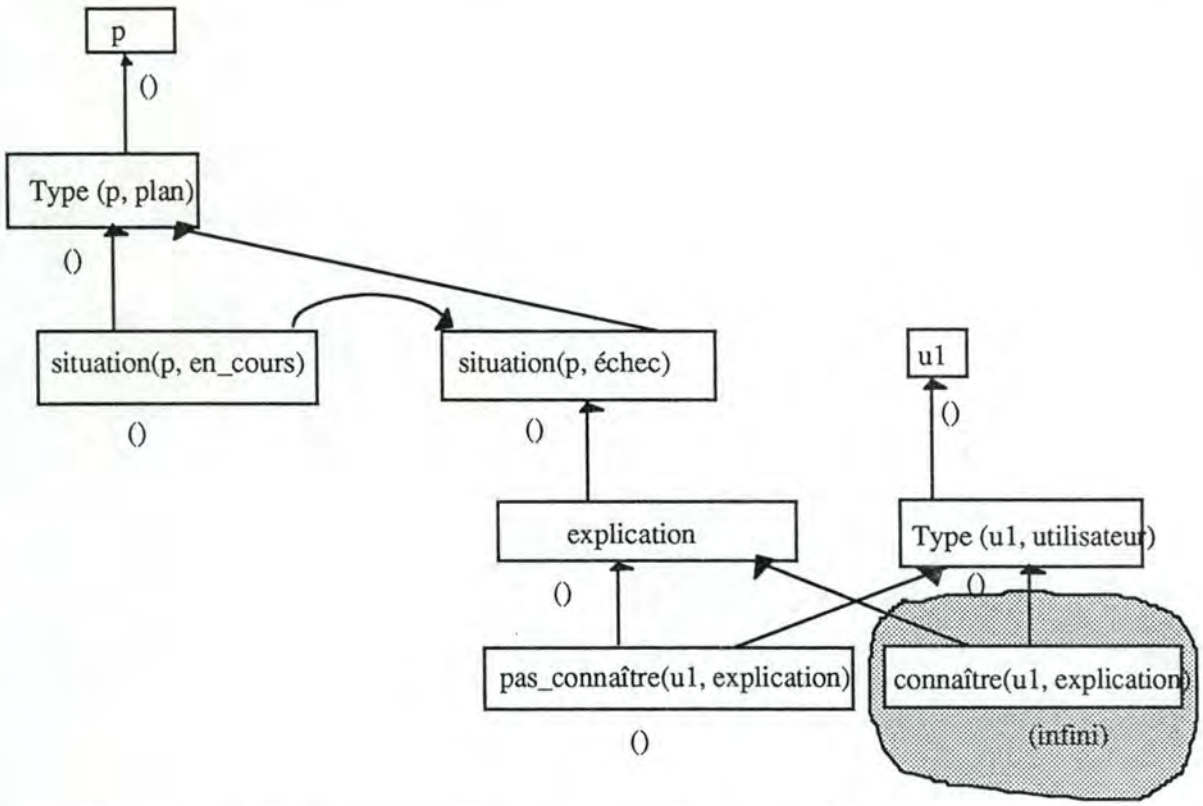
On peut imaginer de présenter l'échec à l'utilisateur en suivant un raisonnement analogue.

Ajoutons à notre modèle, un objet p de type plan :



Dès que la planification échoue et que, pour simplifier[48], l'explication de l'échec est produite, le modèle devient :

[48] On peut imaginer que l'établissement de l'explication de l'échec devienne lui-même un objectif pour la planification. Un plan devient un objet du modèle. Lorsqu'il est dans sa partie "en-cours", on met les différents diagnostics dans des zones incluses. Les relations de précédence et d'inclusion nous permettent de retracer l'arbre de recherche et d'une certaine manière, de faire évoluer nos choix du diagnostic le plus valable. Si la planification réussit, on fait suivre "en-cours" par "réussite", et on exécute le plan. Sinon, on passe à la zone temporelle "échec" à laquelle on inclut une zone à contexte "infini" : explication. On utilise alors des méta opérations sur l'objet plan pour trouver cette explication.



On suppose ici, que le fait d'établir une explication à l'échec a comme corollaire que l'utilisateur y est intéressé et, donc, qu'il ne la connaît pas (ajout de la zone "pas_connaître(u1, explication)") et qu'il veut la connaître (requête implicite).

L'action **Informer**, définie dans le point précédent, permet la réalisation de cette requête.

Actes de langage

L'interface agit donc également sur l'utilisateur. Nous ferons comme hypothèse de départ que ces actes seront traduits en langage; c'est donc tout naturellement que nous nous tournerons vers les actes de langage.

Jusqu'à présent, nous avons étudié ces actes et leurs conséquences dans une optique de reconnaissance du langage. Dans ce qui va suivre, centrons nous sur la problématique inverse : la production de langage.

Informer l'utilisateur

Comme nous l'avons signalé dans les exemples précédents, le **Inform** doit être ajouté à l'ensemble des actions. Il permettra, en l'intégrant au processus de planification ou en lui succédant, de rendre compte à l'utilisateur et/ou de répondre à ses questions.

Sous-dialogues

Doit-on, de la même manière, mettre la requête dans l'ensemble des actions dont dispose l'interface pour planifier? Les travaux de [Allen 80],[Litman 87] laissent penser que oui. Avant de répondre à cette question, examinons les conséquences d'un tel choix.

Pour [Allen 80], la validation d'un état final se fait également par planification. On planifie dans les deux sens (chaînage avant et arrière) jusqu'au moment où il y a blocage. Au lieu de s'arrêter, on ajoute au plan un acte de langage (question) censé lever ce blocage. On anticipe donc la réponse de l'utilisateur. La planification n'échoue jamais.

[Litman 87] propose un mécanisme de gestion de pile de plans pour prendre en compte les écarts non planifiés de l'utilisateur (par exemple : la machine a prévu de demander la couleur à donner à un objet et l'utilisateur lui répond qu'il ne comprend pas ce qu'elle veut dire par "couleur d'objet"). L'utilisateur émet une requête de base, cette requête donne lieu à un plan qui ne peut être complété car il lui manque des informations. A ce premier plan, on en superpose un deuxième qui a comme but d'identifier des paramètres; mais pendant l'exécution de ce plan, l'utilisateur demande à la machine de répéter. On empile alors un plan qui consiste à répéter ce qui vient d'être dit, ... On exécute toujours le plan au sommet de la pile. Lorsque son exécution est complétée, on le dépile; etc.....

[Luzzati 90] se rapproche très fort de ce type de discours. Son modèle, que nous avons proposé au chapitre 3, prétend modéliser et contrôler la profondeur dans le dialogue. Par une approche plus centrée sur le discours plutôt que sur les plans, il met en évidence les mêmes mécanismes complexes.

Notre point de vue est tout autre. [Litman 87], [Luzzati 90] et dans une moindre mesure [Allen 80] ont comme objectif d'arriver à une modélisation de l'évolution de dialogues. Le nôtre est de permettre à un utilisateur d'effectuer une tâche de la façon la plus conviviale possible ([Capindale 90], [Chantraine 90], [Hauptmann 87] nous laissent croire que le LN, même finalisé, le permet). Nous maintenons donc sur notre position : si la planification échoue, on présente les causes de l'échec; l'utilisateur peut laisser tomber ou réessayer avec plus de données. Dans ce cas, on recommence la planification à zéro (on se rapproche du modèle des énoncés maximalistes de [Luzzati 90]).

6.2.4. D'autres actes

[Maybury 92], [Maybury 92 bis] proposent d'étendre le concept d'actes de langage à d'autres modes ou moyens de communication.

Multimédia et multimodal

Commençons par donner une définition de multimédia et multimodal. Pour mettre tout le monde d'accord, nous reprendrons les idées de [Maybury 92] que nous compléterons par [Carbonell 91].

Nous n'attachons pas tellement d'importance à la distinction entre média et mode; mais dans un souci d'éclairer les choses, nous construisons ces définitions autour de ces différences.

Dans une communication Homme-Machine (figure 1.1), on définit le **mode** comme centré sur l'homme et le **médium** comme centré sur la machine.

Par mode, on entend l'utilisation de sens humains pour traiter l'information entrante (vision, audition, touché, ...).

Par médium, on englobe plutôt les moyens physiques (papier, bruit, fumée, ...) et logiques (langage écrit ou parlé, signe, graphiques, ...) par lesquelles l'information transite.

Une feuille hyper-texte permet à la fois une communication multimédia (images, texte et son) et multimodale (vision et audition).

Tentons d'étendre le formalisme des actes de langage aux autres actes de communications.

6.2.5. Communication act

Il est rare que l'homme s'exprime uniquement avec le LN. Les gestes, l'air que l'on se donne, ... sont des moyens (médium) de communication qu'on utilise, en plus du langage, dans nos conversations.

A coté des actes de langage, on trouve donc toute une série d'actes véhiculés par les autres moyens de communication (par exemple les visual act de la figure 6.1 tirée de [Maybury 92]).

Tous ces actes, sont repris sous le titre générique de **communication acts**, et peuvent être utilisés séparément ou de façon combinée.

RHETORICAL ACT	LINGUISTIC ACT	VISUAL ACT
	ILLOCUTIONARY ACT	DEICTIC ACT
identify	inform	highlight, blink, circle, ...
define	request	indicate direction
describe	warm	
detail	concede	DISPLAY CONTROL ACT
divide		display region
illustrate	LOCUTIONARY ACT	zoom (in-out)
compare	assert (declarative)	pan (left, rigth, up, down)
narrate	ask (interrogative)	
explain	command (imperative)	DEPICT ACT
argue	recommend ("should")	depict image
	exclaim (exclamation)	draw (line, arc, circle)

Figure 6.1 . Communicative acts : Rhetorical act, Linguistic, and Visual.

Pour expliquer notre propos, reprenons l'exemple 1. Nous avons supposé qu'il existait une action: **Informer** qui permettait de trouver un plan aux questions de l'utilisateur. Affinons cette action en fonction du moyen de communication qu'elle va utiliser.

On peut, tout d'abord, Informer_par_langage. On définit cette action en utilisant le formalisme du chapitre 3 :

```
Informer_par_langage (interface, utilisateur, fait)

    Pré-condition:

      cando: Believe(interface, fait) & Langagier(fait)

      want:  Believe(interface, Want(interface, inform-
instance))

    Post-condition: Believe(utilisateur, Believe(interface,
fait))

    Décomposition : Convince(interface, utilisateur, fait).
```

La cando_pré-condition oblige l'interface a connaître le fait mais en plus à ce que ce fait soit exprimable sous forme de langage.

La want_pré-condition est respectée à partir du moment où on a ajouté l'objectif au modèle et qu'on a lancé la planification.

Reprenons le Convince pour simplifier la post-condition sans développer la partie décomposition du Convince. C'est tout le problème du choix des mots pour traduire ses intentions (problème de génération de LN). Néanmoins, on peut faire l'hypothèse qu'une affirmation du fait suffise : Assert (interface, utilisateur, fait).

Convince(interface, utilisateur, fait)

Pré-condition:

cando: Believe(utilisateur, Believe(interface, fait))

Post-condition: Believe(utilisateur, fait).

Décomposition : Assert (interface, utilisateur, fait)

L'action peut donc s'écrire :

Informer_par_langage (interface, utilisateur, fait)

Pré-condition:

cando: Believe(interface, fait) & Langagier(fait)

want: Believe(interface, Want(interface, inform-instance))

Post-condition: Believe(utilisateur, fait)

Décomposition : Convince(interface, utilisateur, fait).

De la même façon, on peut envisager de répondre graphiquement en mettant la fenêtre à l'avant plan. L'utilisateur demande la couleur de la fenêtre. On suppose donc qu'il ne peut la voir. On définit alors l'action Informer_graphiquement :

Informer_Graphiquement (interface, utilisateur, fait)

Pré-condition:

cando: Believe(interface, fait) & Visuel(fait) & Appartient(fait, entité)

want: Believe(interface, Want(interface, inform-instance))

Post-condition: Believe(utilisateur, fait)

Décomposition : Make_visible(interface, utilisateur, entité)

L'interface informe l'utilisateur graphiquement d'un fait. Il faut que l'interface connaisse le fait, que le fait soit visuel et qu'il soit une propriété d'une entité sur laquelle on sait agir (cando_pré).

On suppose que l'utilisateur sera informé, si on rend cette entité visible. Pour ce faire, on définit l'action Make_visible :

```
Make_visible (interface, utilisateur, entité)

    Pré-condition:

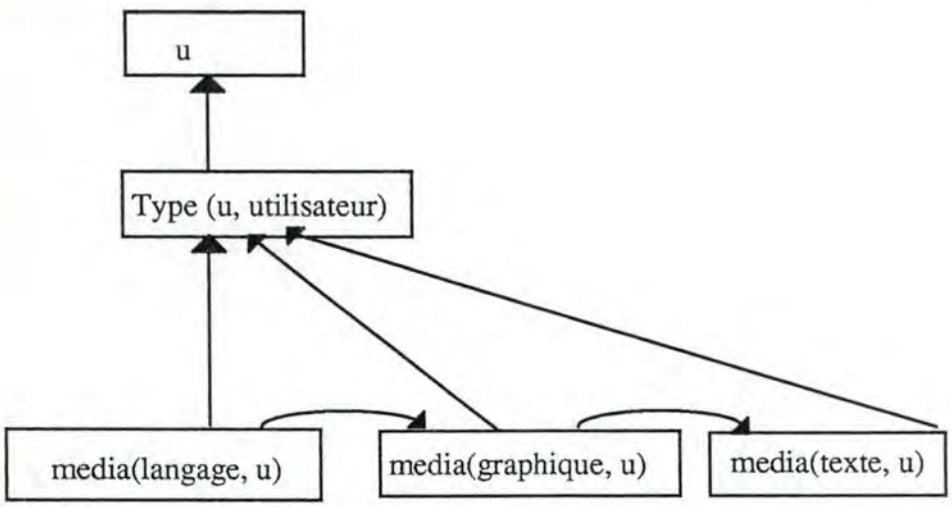
      cando: Displayed(entité)

    Post-condition: Visible(entité)
```

Le planificateur, dans sa recherche, pourra au choix utiliser Informer_par_langage ou Informer_Graphiquement. On peut imaginer toute une série d'autres moyens d'expression qui serviront de support à l'information de l'utilisateur . On peut aussi imaginer une action qui travaillerait sur plusieurs modes à la fois (Informer_Graphiquement_et_par_langage, ...).

Le choix de l'action par le planificateur peut être guidé par les souhaits de l'utilisateur. Si l'interface s'aperçoit que, bien que la fenêtre a été placée à l'avant plan, l'utilisateur réitère sa question, c'est qu'elle a utilisé un mauvais medium par répondre.

Le médium que l'utilisateur préfère peut être modélisé par une ligne de propriété de l'objet utilisateur :



Pour des questions semblables, l'utilisateur u préférerait les réponses en LN, puis sous forme de graphiques; actuellement il est plus réceptif aux textes.

La modélisation de l'utilisateur grâce aux zones temporelles peut permettre également de représenter les habitudes de l'utilisateur à l'intérieur d'un même média. Pour le LN, ce peut être par exemple le fait qu'il préfère telle structure de réponse, à tel temps, ...

Pour plus de détails sur ces actes de communication, on peut se référer à [Maybury 92].

6.2.6. Rhetorical acts

Les Rhetorical acts (figure 6.1) sont, par définition, des actes de communication indépendants du mode ou des média.

Ces actes constituent un méta-niveau aux actes de communication et permettent donc de raisonner, donc de créer des systèmes de dialogue, indépendamment de tout support.

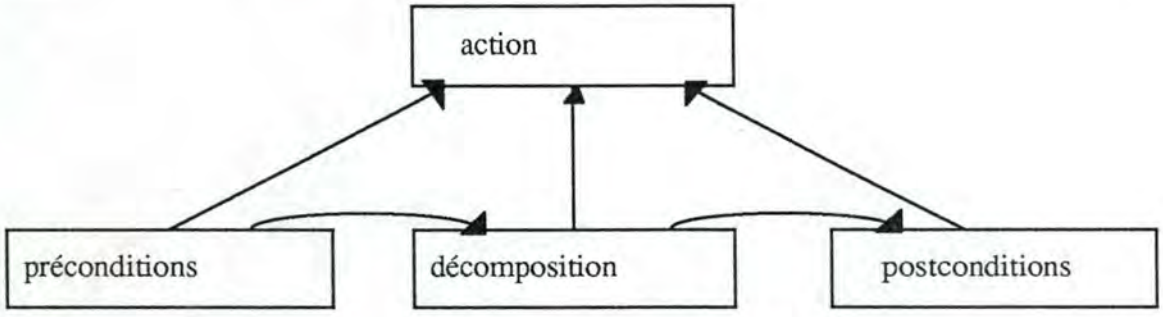
"A strength of rhetorical acts is that they simultaneously distinguish between different forms of communication and integrate common form across medium and mode boundaries by focusing on the semantic content and pragmatic effect of actions wich realize these forms. This permits formalization of these actions as plan operators in a plan library." [Maybury 92 pp. 12].

Ainsi, les rhétorical acts tentent de regrouper les pré et post-conditions communes à un ensemble d'actes de communications. Il va de soi que ces conditions et effets sont indépendants de toutes contraintes quant au mode de communication utilisé et se centrent sur les dispositions mentales de l'émetteur et du récepteur ainsi que sur les conséquences cognitives de la correcte éxécution des tels actes.

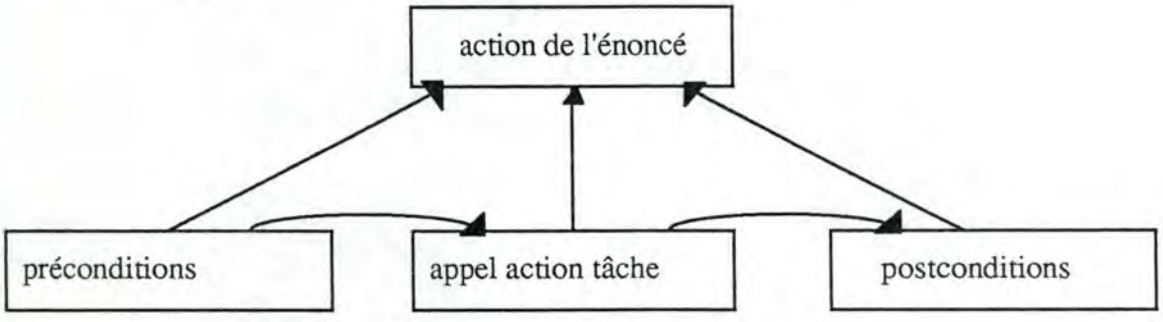
6.3. Agrégation d'actions

Le dernier point que nous survolerons rapidement traitera de la partie décomposition des actions

Nous avons dit qu'une action était représentée dans l'interface d'après le formalisme suivant :

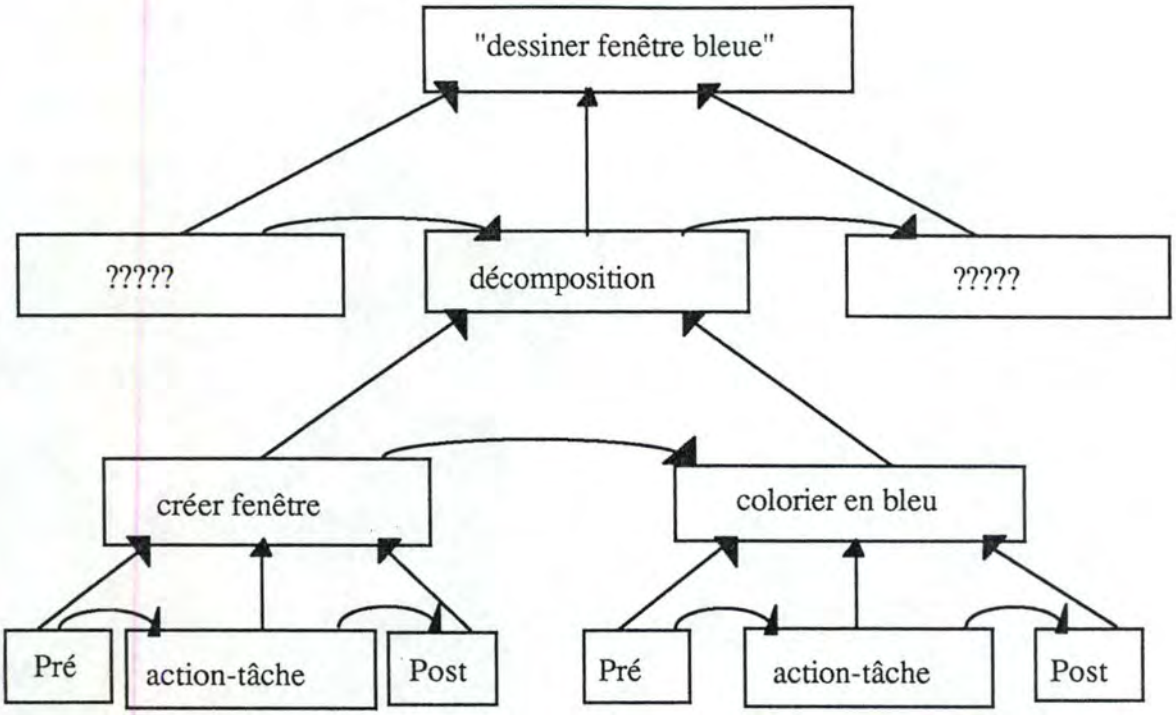


Avant l'implémentation du planificateur, l'interface associait une action de la tâche à l'action comprise dans l'énoncé



La planification permet maintenant d'appliquer des actions qui n'ont pas d'équivalent direct dans la tâche. L'action de l'énoncé est par définition un plan. L'appel de l'action de la tâche est remplacé par la séquence d'actions du plan, qui peuvent elles-mêmes faire l'objet d'une décomposition.

Exemple : le plan "dessiner fenêtre bleue"



On introduit alors une nouvelle sémantique au modèle (Figure 6.2) : la partie décomposition de chaque action est composée d'un ensemble de sous-actions (relation d'inclusion); ces sous-actions se succédant dans le temps (relation d'adjacence entre sous-fait). On décompose jusqu'au moment où on arrive au niveau des appels aux fonctions de la tâche directement exécutables.

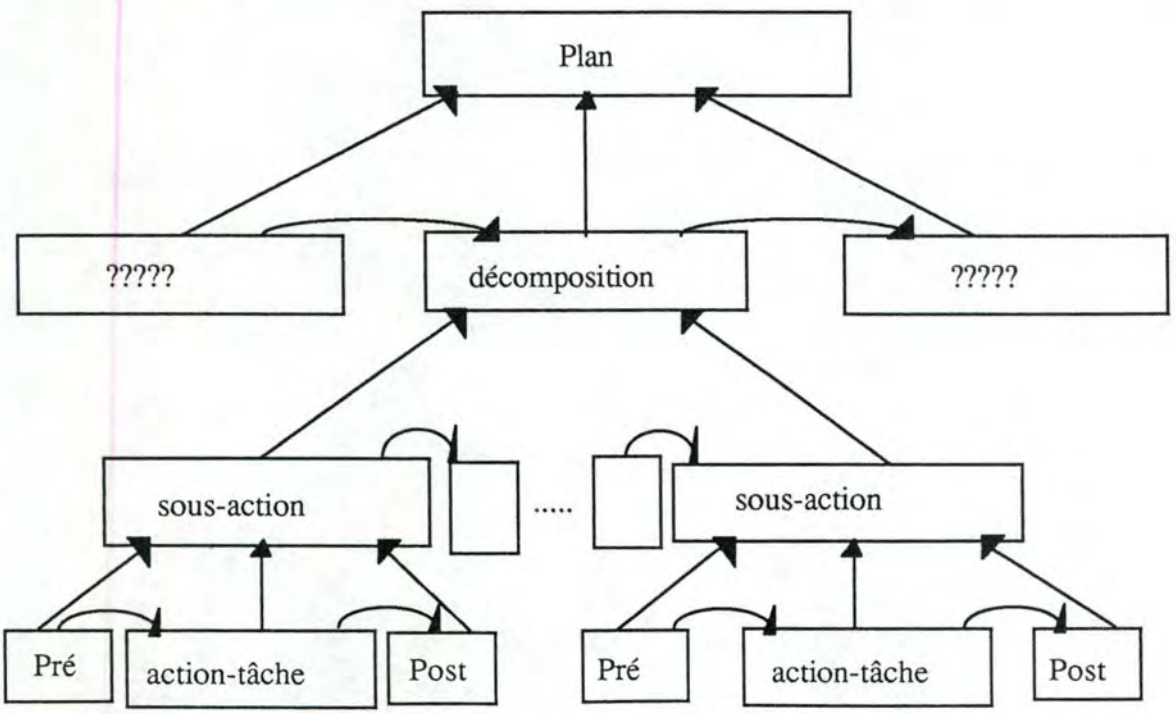


Figure 6.2 : Décomposition d'un plan en actions.

Dès lors, lorsqu'on crée un plan, il est peut-être bon de le reprendre dans la liste d'actions, pour étoffer les connaissances de l'interface. On peut donc imaginer de développer un module d'apprentissage qu'on relierait au planificateur. Un de ses objectifs principaux, sera de définir les pré et post-conditions à l'exécution directe du plan (jusqu'à présent, seulement les pré et post des sous-actions). Ce problème reste lui-aussi ouvert.

Conclusion

La conclusion sera articulée autour du commentaire de la figure 7.1. L'objectif que l'on s'était donné dans l'introduction consistait à rechercher les composantes cognitives nécessaires à l'interface pour devenir "intelligente" (Cfr. figure 0.2).

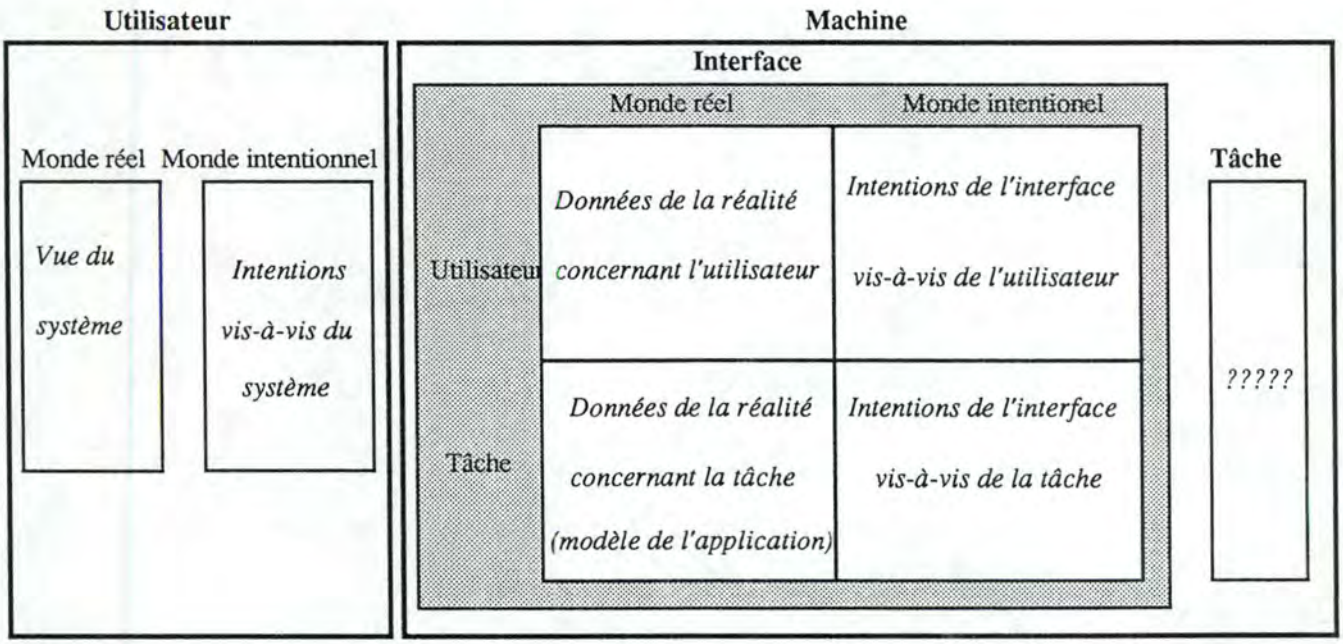


Figure 7.1: Un modèle synthétique de l'interaction.

La réalité qui nous a occupé tout au long de ce mémoire avait trait à l'interaction utilisateur et ordinateur.

Pour les diverses raisons abordées au chapitre 3, nous avons décomposé l'ordinateur en deux centres de traitements : l'interface et la tâche.

Notre objectif étant centré sur le dialogue entre l'utilisateur et l'interface, nous nous sommes surtout intéressés à ces deux entités et avons fait une certaine impasse sur le fonctionnement de la tâche proprement dite (la tâche est perçue comme une boîte noire; on s'intéresse seulement aux messages qu'elle échange avec les autres entités).

En ce qui concerne l'utilisateur, toute modélisation de son comportement n'aura jamais de conséquences qu'a posteriori; on voit mal, en effet, comment on pourrait le re-programmer. Néanmoins, s'il on veut procéder par analogie de comportement pour réaliser une interface "à visage humain", on se doit de modéliser ses connaissances et mécanismes de fonctionnement.

L'utilisateur dispose, entre autres, de connaissances sur l'état de la machine (il ne fait, en principe, pas la différence entre interface et tâche) et d'intentions visant à y apporter des modifications. Ne disposant pas de moyens d'action directs, il transmet ses intentions à l'interface par le langage de commandes. Ceci nous a amené à un des problèmes centraux de ce travail : comment l'interface doit-elle organiser ses connaissances pour permettre le dialogue de commandes et ainsi, la réalisation des intentions de l'utilisateur ?

Un premier type de données englobe les données sur la réalité vue par l'interface :

• Monde réel associé à la tâche : c'est une vision de la tâche par l'interface reprenant l'ensemble des actions possibles sur la tâche et l'état de celle-ci. Cet état doit être le plus proche possible de la réalité, car c'est en fonction de ce modèle que l'interface (et elle seulement) pilote la tâche.

• Monde réel associé à l'utilisateur : reprenant des données physiques sur l'utilisateur pour permettre un dialogue convivial (nom, présence devant écran, ...) et des données sur la tâche qu'il connaît (objet visible ou pas, ...) ou ne connaît pas.

Le second type de données regroupe les intentions de l'interface sur la réalité :

• Monde intentionnel associé à la tâche : le dialogue de commandes entre l'utilisateur et l'interface "contient" des actes de langage de requête dont l'exécution a comme conséquence que les intentions de l'utilisateur vis-à-vis d'une tâche deviennent celles de l'interface.

• Monde intentionnel associé à l'utilisateur : en principe, l'utilisateur est seulement en contact avec l'interface. Lorsque celle-ci modifie la tâche, elle doit ensuite se fixer comme objectif d'en informer l'utilisateur (pour qu'il puisse actualiser sa représentation du monde réel). Si, par contre, elle ne parvient pas à agir sur la tâche, elle doit quand même informer l'utilisateur de cet échec (négociation).

La planification part des informations contenues dans les mondes réel et intentionnel associés à la tâche et cherche à réaliser ces objectifs :

En cas de succès, l'interface modifie son monde réel et envoie la séquence d'actions à la tâche qui, en retour, valide la nouvelle représentation de la réalité.

En cas d'échec, le planificateur doit permettre de l'expliquer.

La planification part des informations contenues dans les mondes réel et intentionnel associés à l'utilisateur.

Il ne s'agit plus ici de planification à proprement parler mais plutôt de choix du meilleur mode d'expression, compte tenu de la position de l'utilisateur et du contenu informationnel à transmettre. L'objectif à réaliser par l'interface est toujours d'informer l'utilisateur d'une situation. Cette situation peut être une réponse à une requête qui ne demandait pas d'action sur la tâche (question sur l'état de la tâche, par exemple), une présentation d'un nouvel état de la tâche modifié par l'exécution du plan, la présentation des causes de l'échec de l'interprétation des énoncés de l'utilisateur et plus précisément de la planification, ... Il est clair que cette "planification" ne doit jamais échouer, sous peine de stopper le dialogue coopératif entre l'interface et l'utilisateur

L'utilisation des zones temporelles et de la sémantique associée, pour modéliser les mondes réels et intentionnels associés à la tâche et à l'utilisateur dans l'interface permet la cohérence entre ces modèles, la cohérence entre ce niveau de recherche et les niveaux inférieurs (utilisation du même modèle pour la recherche des référents), l'écriture d'un planificateur dont les résultats sont directement inscrits sous forme de ZT, la possibilité de traiter les requêtes orientées "moyens" ou "résultat" en suivant la même logique, des facilités de constitution de diagnostics d'échec de planification et, l'instauration d'une transparence et d'une complémentarité entre les modes de communication utilisés dans le dialogue.

Perspectives

Les perspectives de recherche sont nombreuses, tant le sujet paraît ouvert et propice à des essais d'implémentation plus concrets, plus généraux et seuls capables de guider une mise au point optimale (notamment en ce qui concerne les heuristiques qui doivent être utilisés pour la planification).

Les premières recherches qui, à notre avis, doivent être poursuivies sont celles qui concernent la modélisation de la tâche et tout particulièrement la définition de mécanismes sur les ZT associés à des suppressions d'objets de la tâche.

Un deuxième pôle d'intérêt concerne le planificateur lui-même. Il faudrait dans un premier temps, étendre la planification à la recherche de toutes les séquences d'actions possibles pour des objectifs donnés, et régler ensuite le problème du choix du "meilleur" plan. Le problème du choix se pose aussi lorsqu'on s'attaque à la recherche du meilleur diagnostic pour expliquer l'échec.

Un troisième centre d'intérêt, fortement lié au deuxième concerne plus la nature de la finalité de la requête. L'utilisateur veut-il que la machine réalise ses intentions comme bon lui semble ou d'une certaine façon ? La finalité de la requête porte-t-elle sur le résultat ou les moyens d'y arriver ?

Enfin, la conception d'interface doit suivre la tendance "multimodale"; un développement des concepts introduits au chapitre 6 et leur traduction sous forme de ZT semble tout à fait naturel et, si on peut encore en douter, devra mettre encore une fois en évidence la souplesse du modèle.

Bibliographie

[Allen] Allen J. & Perrault R., Analysing intention in utterances, in : *Artificial Intelligence* 15, pp 143-178.

[Andre 92] Andre Elisabeth & Rist Thomas, The design of Illustrated Documents as a Planning Task, Actes de conférence de la *Fourth European Summer School In Logic, Language and Information*; August 17-29 1992; University of Essex; Colchester U.K.

[Antonacci 92] Antonacci F. & Calamani C., Capturing the deep meaning of texts through deduction and inference, in : *IBM J. Res. Develop.* Vol. 36 No 3, May 1992.

[Beadon 91] Beardon C., Lumsden D., Holmes G., *Natural langage and computational linguistic*, 1991.

[Bodart 89] Bodart F., Pigneur Y., *Conception des systèmes d'information. Méthodes, Modèles, Outils*. Masson, 1989.

[Bodart 90] Bodart F., notes provisoire du cours d'interface homme-machine, chapitre introductif, FUNDP Namur, Belgique, 1989.

[Buchheit 90] Buchheit Paul & Moher Thomas, Response assertiveness in human-computer dialogues, *Int. J. Man-Machine Studies* 32, pp 109-117, 1990.

[Capindale 90] Capindale Ruth A. & Crawford Robert G., Using a natural language interface with casual users, *Int. J. Man-Machine Studies* 32, pp 341-362, 1990.

[Carbonell 91] Romary L., Carbonell N. & Pierrel J.M., *Multimodal Human-Computer Dialogue*, CRIN 91-R-113, 1991.

[Carlson 85] Carlson Lauri, *Dialogue games*, 1985.

[Chantraine 90] Chantraine Y. et Hupet M., Le modèle collaboratif de la référence : La contribution de l'interlocuteur est-elle crutiale?, *actes du colloque de L'ARC*, Paris 1990.

[Charniak 76] Charniak E., *Computational semantics*, 1976.

[Cohen] Cohen P. & Perrault R., Elements of a plan based theory of speech, in *Cognitive Science* 3, pp 117-212.

[Deville 89] Deville Guy, *Modelization of task-oriented utterances in a Man-Machine Dialogue system,* Proefschrift ter verkrijging van de graad van Doctor in de Letteren en Wijsbegeerte aan de Univeristaire Instelling Antwerpen te vededigen door, 1989.

[Doe 92] Doe G.J., Ringland G.A. & Wilson M.D. , A meaning Representation Language for Co-operative Dialogue, actes de conférence de la *Fourth European Summer School In Loginc, Language and Information*; August 17-29 1992; University of Essex; Colchester U.K.

[Enjalbert 90] Enjalbert P., "Savoir qui, quand, où ..." Une analyse centrée sur les notions de question et de contexte de résolution de problème, *actes du colloque de L'ARC*, Paris 1990.

[Fox 87] Fox B.A., Interactional reconstruction in real-time language processing, in *Cognitive science* 11, 1987 pp 365-387.

[Gaiffe 90] Gaiffe B. & Romary L., *Managing multimodal information and references in the Multiworks project*, CRIN 90-R-168, 1990.

[Haton 91] Haton J.P., Pierrel J.M., Perennou G., Coelen J., Gauvain J.L.; *Reconnaissance automatique de la parole*, 1991.

[Haton 92] Haton J.P. & co-auteurs, *Reconnaissance automatique de la parole*, 1992.

[Hauptmann 88] Hauptmann A.G. & Rudnicky A.I., Talking to computers: an empirical investigation, *Int. J. Man-Machine Studies* 28, pp 583-604, 1988.

[Klein 90] Klein J. & Pierrel J.M., *Interprétation d'expresions complexes dans un système de dialogue homme-machine*, CRIN 90-R-178, 1990.

[Levinson 83] Levinson Stephen C., *Pragmatics*, Cambridge University Press,1983.

[Litman 87] Litman D.J. & Allen, A plan regognition model for subdialogues in Conversation, in *Cognitive science* 11, 1987 pp 163-200.

[Luzzati 90] Luzzati D., Un modèle de dialogue pour la gestion de l'interaction verbale homme machine, *actes du colloque de L'ARC*, Paris 1990.

[Martin 89] Martin G. L., The utility of speech input in user-computer interfaces, *Int. J. Man-Machine Studies* 30, pp 355-375, 1989.

[Maybury 92 bis] Maybury Mark T. , Planning Multimedia Explanations Using Communicative Acts, actes de conférence de la *Fourth European Summer School In Logic, Language and Information*; August 17-29 1992; University of Essex; Colchester U.K.

[Maybury 92] Maybury Mark T., Communicative Acts for Multimedia and Multimodal Dialogue,actes de conférence de la *Fourth European Summer School In Logic, Language and Information*; August 17-29 1992; University of Essex; Colchester U.K.

[Mignot 92] Mignot Christophe, Rapport interne mensuel, CRIN 1992.

[Moeschler 89] Moeschler J., *Modélisation du dialogue, représentation de l'inférence argumentative*, 1989.

[Perrault 92] Perrault & Wilensky, Chapter 4 : Discourse theory, goal & speech acts; , in *Theoretical Issues in natural language processing*, Y. Wilks editor, 1992.

[Pierrel 87] Pierrel J.M., *Dialogue oral homme machine*, Edition Hermes, Paris, 1987.

[Romary 89] Romary L., *Vers la définition d'un modèle cognitif autour de la représentation du temps dans un système de dialogue homme-machine*, Thèse de l'universite de Nancy 1, 1989.

[Romary 90] Romary L. & Pierrel J.M., *Perception, Langage, Raisonnement : Une même représentation temporelle*, CRIN 90-R-054, 1990.

[Romary 91] Romary L., Integration of spatial and temporal information produced by a natural language discourse, *KMET 1991*, Sophia Antipolis, France, IOS Press, 1991.

[Sabah 89] Sabah G., *L'I.A. et le langage*, volume 1 et 2, 1989.

[Sadek 87] Sadek M., Guyomard M., Siroux J., "Vers un système de dialogue basé sur les logiques intentionnelles et la planification des actes de langages", *Congrès AFCET-RFIA*, Antibes, Nov. 1987.

[Scapin] Scapin D.L., *Guide ergonomique de conception des interfaces homme-ordinateur*, INRIA Rocquencourt, France.

[Wetter 92] Wetter T. & Nüse R., Use of natural language for knowledge acquisition: Strategies to cope with semantic and pragmatic variation, in *IBM J. Res. Develop.* Vol. 36 No 3, May 1992.

[Zoltan 91] Zoltan-Ford Elisabeth, How to get people to say and type what computers can understand, *Int. J. Man-Machine Studies* 34, pp 327-347, 1991.

Pour de futurs développements :

§ [Carberry 83] Carberry M.S. Tracking users goals in an information-seeking environment, in *Proceedings of the national coference on artificial intelligence*, 1983, Washington, D.C.

§ [Cohen 78] Cohen P.R. *On Knowing What to say: Planning Speech Acts*, Ph.D. Thesis, University of Toronto, 1978.

§ [Grosz 86] Grosz B.J. & Sidner C.L. Attention, Intentions and the structure of Discourse, in *Computational Linguistic* 12, pp 175-204 , 1986.

§ [Kautz 85] Kautz H.A. Towards a theory of plan recognition. *Tech.Rep.N°170*, Rochester, N.Y: University of Rochester, 1985.

§ [Kautz 86] Kautz H.A. & Allen J.F. Generalised plan recognition, in P*roceedings of the National conference on Artificial Intelligence*, Philadelphia, P.A 1986.

§ [Litman 89] Litman D.J. *Plan recognition and Discourse analysis: An integrated approach for understanding Dialogues* (tech Rep. N° 170 and PhD Thesis), University of Rochester, N.Y.

§ [Pollack 86] Pollack M.E. A model of plan inference that distinguishes between the believes of actors and observers, in *Proceedings of the meeting of the association for Computational LInguistics*, New York, N.Y, 1986.

§ [Sidner 85] Sidner C.L. Plan parsing for intended response recognition in discourse , in *Computational Intelligence* 1 , pp 1-10,1985.

§ [Wilensky 83] Wilensky R., Planning and Understanding : A computational Approach to Human Reasoning, 1983, Addisson-Wesley, N.Y.